SOLUTIONS MANUAL FOR

Discovering Computer Science: Interdisciplinary Problems, Principles, and Python Programming

by

Jessen Havill

CRC Press
Taylor & Francis Group

SOLUTIONS MANUAL FOR

Discovering Computer Science: Interdisciplinary Problems, Principles, and Python Programming

by

Jessen Havill

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper
Version Date: 20150527

International Standard Book Number-13: 978-1-4822-5422-8 (Ancillary)

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Discovering Computer Science

Interdisciplinary Problems, Principles, and Python Programming

Solutions Manual

Jessen Havill

January 4, 2016

Contents

CHAPTER 1

What is computation?

1.1 EXERCISE SOLUTIONS

Section 1.2

1.2.2 **Algorithm** Minimum Value

Input: a list of numbers

(a) Initialize the current minimum value to the first number in the list.

(b) For each additional number n in the list, repeat the following:

If n is less than our current minimum value, then remember n as our new minimum value.

Output: the current minimum value

1.2.3 **Algorithm** Cookies

Input: a list of y's and o's

1. Initialize the running sum to 0.
2. For each letter in the list, repeat the following:
   (a) If the letter is y, then add 2 to the running sum, and assign the result to be the new running sum.
   (b) Otherwise, if the letter is o, then add 3 to the running sum, and assign the result to be the new running sum.

Output: the value of the running sum

Section 1.3

1.3.1 Yes, it is possible. Follow the algorithm in part (c) of the figure, but have person A call person G during time 3.

1.3.2 (a) There are 5 calls made during time step 4, 8 during time step 5, and 13 during time step 6.

(b) In general, you can determine the number of calls made during time step t by adding the number of calls made during time steps $t-1$ and $t-2$. (This sequence is called the Fibonacci sequence.)

1.3.3 The most important criterion is the algorithm's time complexity, which is how quickly the algorithm can compute its output with respect to the size of the input. Other criteria might be the algorithm's compactness, the amount of memory it requires, or how clearly the algorithm is expressed.

1.3.4 The algorithm requires $n+1$ arithmetic operations, so its asymptotic time complexity is n.

Section 1.4

1.4.1 The binary number `1101` is equivalent to $1 + 4 + 8 = 13$ in decimal.

1.4.2 The binary number `1111101000` is equivalent to $512 + 256 + 128 + 64 + 32 + 8 = 1,000$ in decimal.

1.4.3 The binary number `11.0011` is equivalent to $2 + 1 + 1/8 + 1/16 = 33/16$ in decimal.

1.4.4 The binary number `11.110001` is equivalent to $2 + 1 + 1/2 + 1/4 + 1/64 = 349/64$ in decimal.

1.4.5 The decimal number $22 = 16 + 4 + 2$ is equivalent to `10110` in binary.

1.4.6 The decimal number $222 = 128 + 64 + 16 + 8 + 4 + 2$ is equivalent to `11011110` in binary.

1.4.7 The decimal number 0.1 is approximately `0.000110` in binary. This binary number is actually equivalent to $1/16 + 1/32 = 3/32 = 0.09375$ in decimal.

1.4.8 The image could be stored row by row (called "row major order"):

```
011100
010010
010001
010001
010010
011100
```

or column by column (called "column major order"):

```
000000
111111
100001
100001
010010
001100
```

1.4.9

a	b	a **and** b	**not** (a **and** b)
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

1.4.10

a	b	**not** a	**not** b	**not** a **or not** b
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

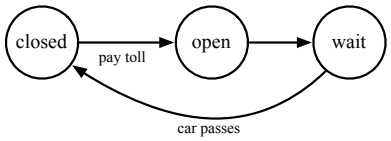
Notice that **not** a **or not** b = **not** (a **and** b). This is one of *De Morgan's laws*; the other is the solution to the next problem.

1.4.11

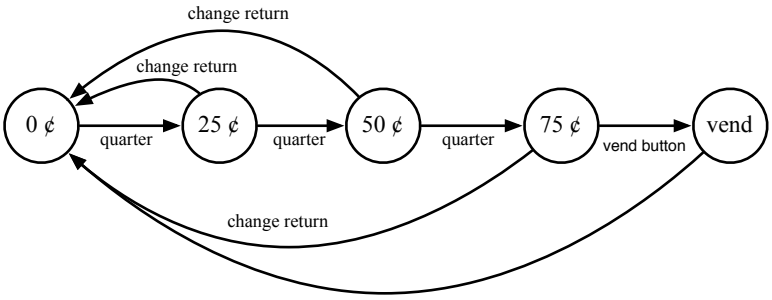
a	b	**not** a	**not** b	**not** a **and not** b
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

Notice that **not** a **and not** b = **not** (a **or** b). This is one of *De Morgan's Laws*; the other is the solution to the previous problem.

1.4.12



1.4.13


change return
change return
0 ¢
quarter
25 ¢
quarter
50 ¢
quarter
75 ¢
vend button
vend
change return

CHAPTER 2

Elementary computations

2.1 EXERCISE SOLUTIONS

Section 2.2

2.2.1
```
838 * 5280 * 12
```

2.2.2
```
0.05 * 2 ** 24
0.05 * 1.5 ** 24
```

2.2.3
```
838.0 / (32.6 * 2)
```

2.2.4
```
2e5 / 4.54e9
```

2.2.5
```
4.54e9 / 60 / 60 / 24 / 365   # 143.96245560629123 years
```

2.2.6
```
4.54e9 / 2.8e9
```

2.2.7
```
2 ** 40 / (8 * 2 ** 20)
```

2.2.8 (a) 43

(b) 9

(c) 22

(d) 22.5

(e) 1

(f) 22.5

Section 2.3

2.3.1
```
dnaCell = 3.4e-10 * 6e9
dnaBody = dnaCell * 50e12
distSun = 1.49598e8 * 1e3
roundTrips = dnaBody / (distSun * 2)
```

2.3.2
```
radius = 10
area = 3.141592 * radius ** 2
```

2.3.3 `area` remains the same because it was not assigned a new value.

2.3.4 The value of `x` is overwritten by the value of `y` in the first statement. So the second statement does nothing useful. A correct implementation is:

```
temp = x
x = y
y = temp
```

2.3.5 x is 6, y is 24.0

2.3.6 x is 3

2.3.7 x is 12.0 and y is 72.0

2.3.8
```
ones = x % 10
tens = (x % 100) // 10
hundreds = (x % 1000) // 100
```

Section 2.4

2.4.1 The result has the same type as the argument.

2.4.2
```
>>> radius = 10
>>> area = math.pi * radius ** 2
```

2.4.3
```
>>> math.sqrt(18 * 31)
```

2.4.4
```
>>> P = 10000
>>> r = 0.01
>>> n = 12
>>> t = 10
>>> A1 = P * (1 + r / n) ** (n * t)
>>> A2 = P * math.exp(r * t)
>>> A2 - A1
0.460222633122612
```

2.4.5
```
>>> x = 3.1415926
>>> x = int(x * 100) / 100
```

2.4.6
```
>>> x = x - int(x)
```

2.4.7
```
a = 3
b = 4
c = -5

x1 = (-b + math.sqrt(b ** 2 - 4 * a * c)) / (2 * a)
x2 = (-b - math.sqrt(b ** 2 - 4 * a * c)) / (2 * a)
```

2.4.8
```
distance = math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)
```

2.4.9
```
volume = a * b * c * math.sqrt(1 + 2 * math.cos(x) * math.cos(y) * math.cos(z)
                               - math.cos(x)**2 - math.cos(y)**2 - math.cos(z)**2)
```

2.4.10
```
volume = a * b * c * math.sqrt(1 + 2 * math.cos(math.radians(x))
                                   * math.cos(math.radians(y))
                                   * math.cos(math.radians(z))
                                 - math.cos(math.radians(x))**2
                                 - math.cos(math.radians(y))**2
                                 - math.cos(math.radians(z))**2)
```

2.4.11
```
P = float(input('Principal: '))
r = float(input('Interest rate: '))
t = int(input('Number of years: '))
n = int(input('Number of times compounded per year: '))
A1 = P * (1 + r / n) ** (n * t)
A2 = P * math.exp(r * t)
print('You would have \$' + str(A2 - A1) + ' more.')
```

2.4.12
```
a = float(input('Value of a: '))
b = float(input('Value of b: '))
c = float(input('Value of c: '))
```

```
x1 = (-b + math.sqrt(b ** 2 - 4 * a * c)) / (2 * a)
x2 = (-b - math.sqrt(b ** 2 - 4 * a * c)) / (2 * a)
print('The two solutions are', x1, 'and', x2)
```

Section 2.5

2.5.1

```
  0 0 1 0 0 1
+ 0 0 1 1 0 1
-------------
  0 1 0 1 1 0
```

2.5.2

```
  0 0 0 1 0 1 0
+ 0 1 0 1 1 0 1
---------------
  0 1 1 0 1 1 1
```

2.5.3

```
  1 0 0 1
+ 1 1 0 1
---------
  0 1 1 0
```

This answer is incorrect since we had to discard the leftmost `1` from the answer.

2.5.4

```
  0 0 1 0 1 0
+ 1 0 1 1 0 1
-------------
  1 1 0 1 1 1
```

This answer is correct since we did not have to discard a leftmost `1` from the answer.

2.5.5 You can tell that the answer is incorrect if it is smaller than the largest operand.

2.5.6

```
  0 1 0 1
+ 1 1 0 1
---------
  0 0 1 0
```

This answer is correct: $5 + -3 = 2$.

2.5.7 The largest positive number is `0111`, which is 7 in decimal. The smallest negative number is `1000`, which is -8 in decimal.

2.5.8

```
  1 0 0 1
+ 1 1 0 1
---------
  0 1 1 0
```

This answer is incorrect: $-7 + -3 \neq 3$. This occurred because the desired answer, -10, does not fit in four bits.

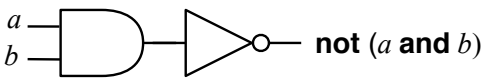
2.5.9

```
  0 0 1 0 1 0
+ 1 0 1 1 0 1
-------------
  1 1 0 1 1 1
```

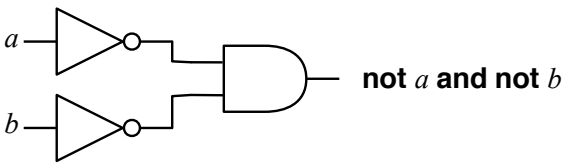
This answer is correct: $10 + -19 = -9$.

2.5.10 You can tell that the answer is incorrect if it has an incorrect sign. This can only happen when we add two positive integers or two negative integers.

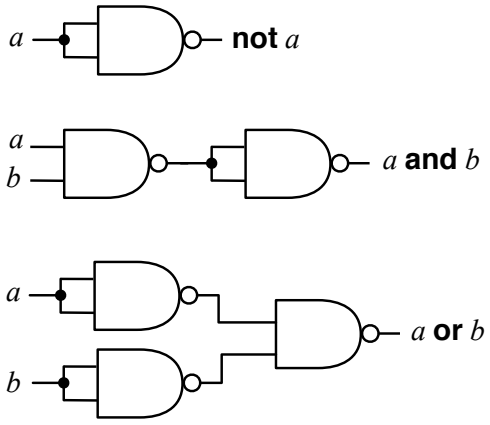
2.5.14



2.5.15



2.5.16



CHAPTER 3

Visualizing abstraction

3.1 EXERCISE SOLUTIONS

Section 3.1

3.1.1 An ADT is a description of a category of data that includes attributes and operations. A class is used to implement an ADT.

3.1.3 Hiding the attributes of an object prevents a program from changing their values in harmful ways. By only changing the values of attributes via methods, we ensure that they are only changed in ways that make sense for the class. Hiding the attributes also makes it possible to change the implementation of the class at a later time without breaking any programs that use it.

Section 3.2

3.2.1
```
import turtle

blueTurtle = turtle.Turtle()
blueTurtle.color('blue')
blueTurtle.forward(75)
blueTurtle.left(45)
blueTurtle.forward(100)

redTurtle = turtle.Turtle()
redTurtle.color('red')
redTurtle.left(120)
redTurtle.forward(100)
redTurtle.left(90)
redTurtle.forward(50)
```

3.2.2 The program changes the for loop to the following:

```
for segment in range(18):
    george.forward(200)
    george.left(100)
```

3.2.3
```
import turtle

george = turtle.Turtle()
for side in range(4):
    george.forward(200)
```

```
    george.left(90)
screen = george.getscreen()
screen.exitonclick()
```

3.2.4
```
import turtle

george = turtle.Turtle()
for side in range(2):
    george.forward(200)
    george.left(90)
    george.forward(100)
    george.left(90)
screen = george.getscreen()
screen.exitonclick()
```

3.2.5 This program draws a filled in 'D'.

```
import turtle

george = turtle.Turtle()      # create a turtle
george.setposition(0, 100)    # move to starting position

# draw a red filled outer semicircle
george.pencolor('red')        # set the turtle tail to red
george.fillcolor('red')       # set the fill color to red
george.begin_fill()
george.circle(-100, 180)      # draw 180 degrees of a circle with radius 100
                              # (negative causes turtle to draw to right)
george.right(90)              # turn 90 degrees and draw a straight line down
george.forward(200)
george.end_fill()

george.up()                   # lift turtle's tail and move for inner semicircle
george.right(90)
george.forward(25)
george.right(90)
george.forward(50)
george.left(90)
george.down()
george.pencolor('white')      # draw a white filled inner semicircle
george.fillcolor('white')
george.begin_fill()
george.circle(-50, 180)
george.right(90)
george.forward(100)
george.end_fill()

george.hideturtle()           # hide the turtle so it doesn't mess up picture
screen = george.getscreen()
screen.exitonclick()          # wait for a mouse click to close window
```

Section 3.3

3.3.1
```
def bloom(tortoise, fcolor, length, petals):
    '''Draws a bloom with the given number of petals.
       petals should not be a multiple of 3.
```

```
    '''

    tortoise.pencolor('red')
    tortoise.fillcolor(fcolor)
    tortoise.begin_fill()
    for segment in range(petals):
        tortoise.forward(length)
        tortoise.left(1080/petals)
    tortoise.end_fill()
```

3.3.2

```
import turtle

def bloom(tortoise, fcolor, length):
    tortoise.pencolor('red')
    tortoise.fillcolor(fcolor)
    tortoise.begin_fill()
    for segment in range(8):
        tortoise.forward(length)
        tortoise.left(135)
    tortoise.end_fill()

def stem(tortoise, length):
    tortoise.pencolor('green')
    tortoise.pensize(length / 20)
    tortoise.up()
    tortoise.forward(length / 2)
    tortoise.down()
    tortoise.right(90)
    tortoise.forward(length)

def flower(tortoise, fcolor1, fcolor2, length):
    bloom(tortoise, fcolor1, length)
    tortoise.up()
    tortoise.left(22.5)
    tortoise.forward(length * 1.082387 / 4)
    tortoise.right(22.5)
    tortoise.down()
    bloom(tortoise, fcolor2, length / 2)
    stem(tortoise, length / 2)
    tortoise.forward(length / 2)

def main():
    george = turtle.Turtle()
    george.hideturtle()
    george.speed(6)
    flower(george, 'yellow', 'red', 200)
    screen = george.getscreen()
    screen.exitonclick()

main()
```

3.3.3

```
import turtle

george = turtle.Turtle()
george.circle(50, -180)  # C
```

```
george.up()              # move to next letter
george.goto(100, 100)
george.down()
george.circle(50)        # O
george.up()              # move to next letter
george.goto(200, 0)
george.down()
george.right(90)         # D
george.forward(100)
george.left(90)
george.circle(50, -180)
george.up()              # move to next letter
george.goto(300, 0)
george.down()
george.left(90)          # E
george.forward(100)
george.right(90)
george.forward(50)
for line in range(2):
    george.backward(50)
    george.left(90)
    george.backward(50)
    george.right(90)
    george.forward(50)
screen = george.getscreen()
screen.exitonclick()
```

3.3.4

```
import turtle

def C():
    george.circle(50, -180)  # C

def O():
    george.circle(50)        # O

def D():
    george.right(90)         # D
    george.forward(100)
    george.left(90)
    george.circle(50, -180)

def E():
    george.left(90)          # E
    george.forward(100)
    george.right(90)
    george.forward(50)
    for line in range(2):
        george.backward(50)
        george.left(90)
        george.backward(50)
        george.right(90)
        george.forward(50)

george = turtle.Turtle()
george.left(180)
```

```
D()
george.up()              # move to next letter
george.goto(100, 0)
george.down()
E()
george.up()              # move to next letter
george.goto(250, 0)
george.down()
C()
george.up()              # move to next letter
george.goto(350, 100)
george.down()
O()
george.up()              # move to next letter
george.goto(450, 0)
george.down()
D()
george.up()              # move to next letter
george.goto(550, 0)
george.down()
E()
screen = george.getscreen()
screen.exitonclick()
```

3.3.5
```
def drawSquare(tortoise, width):
    for side in range(4):
        tortoise.forward(width)
        tortoise.right(90)
```

3.3.6
```
def drawRectangle(tortoise, length, width):
    for side in range(2):
        tortoise.forward(length)
        tortoise.right(90)
        tortoise.forward(width)
        tortoise.right(90)
```

3.3.7
```
def drawPolygon(tortoise, sideLength, numSides):
    turnAngle = 360 / numSides
    for i in range(numSides):
        tortoise.forward(sideLength)
        tortoise.right(turnAngle)
```

3.3.8
```
def drawCircle(tortoise, radius):
    circumference = 2 * 3.1415 * radius
    sideLength = circumference / 360
    drawPolygon(tortoise, sideLength, 360)
```

3.3.9
```
def horizontalCircles(tortoise):
    tortoise.up()
    tortoise.backward(500)
    tortoise.down()
    for numCircles in range(10):
        tortoise.up()
        tortoise.forward(100)
        tortoise.down()
        tortoise.circle(50)
```

3.3.10
```
def diagonalCircles(tortoise):
```

```
        tortoise.up()
        tortoise.backward(500)
        tortoise.right(90)
        tortoise.backward(500)
        tortoise.left(90)
        tortoise.down()
        for numCircles in range(10):
            tortoise.up()
            tortoise.forward(100)
            tortoise.right(90)
            tortoise.forward(100)
            tortoise.left(90)
            tortoise.down()
            tortoise.circle(50)
```

3.3.11
```
def drawRow(tortoise):
    for square in range(4):
        tortoise.fillcolor('red')
        tortoise.begin_fill()
        drawSquare(tortoise, 100)
        tortoise.end_fill()
        tortoise.forward(100)
        tortoise.fillcolor('black')
        tortoise.begin_fill()
        drawSquare(tortoise, 100)
        tortoise.end_fill()
        tortoise.forward(100)
```

3.3.12
```
def drawRow(tortoise, color1, color2):
    for square in range(4):
        tortoise.fillcolor(color1)
        tortoise.begin_fill()
        drawSquare(tortoise, 100)
        tortoise.end_fill()
        tortoise.forward(100)
        tortoise.fillcolor(color2)
        tortoise.begin_fill()
        drawSquare(tortoise, 100)
        tortoise.end_fill()
        tortoise.forward(100)
```

3.3.13
```
def checkerBoard(tortoise):
    tortoise.up()
    tortoise.backward(400)
    tortoise.right(90)
    tortoise.backward(400)
    tortoise.left(90)
    tortoise.down()

    for row in range(4):
        drawRow(tortoise, 'red', 'black')

        tortoise.up()
        tortoise.backward(800)
        tortoise.right(90)
        tortoise.forward(100)
```

```
            tortoise.left(90)
            tortoise.down()

            drawRow(tortoise, 'black', 'red')

            tortoise.up()
            tortoise.backward(800)
            tortoise.right(90)
            tortoise.forward(100)
            tortoise.left(90)
            tortoise.down()
```

3.3.14
```
def polyFlower(tortoise, sideLength, numSides, numPolygons):
    turnAngle = 360 / numPolygons
    for i in range(numPolygons):
        drawPolygon(tortoise, sideLength, numSides)
        tortoise.right(turnAngle)
```

3.3.15
```
def randomWalk(steps):
    tortoise = turtle.Turtle()
    tortoise.pencolor('blue')
    tortoise.speed(0)

    for m in range(steps):
        angle = random.randrange(360)
        tortoise.setheading(angle)
        tortoise.forward(10)

    screen = tortoise.getscreen()
    screen.exitonclick()
```

3.3.16
```
def basketball(fieldGoals, threePointers)
    points = 2 * fieldGoals + 3 * threePointers
    print('The score is', points)
```

3.3.17
```
def age(birthYear):
    print(2014 - birthYear - 1)
```

3.3.18
```
def cheer(teamName):
    print('Go', teamName)
```

3.3.19
```
def sum(number1, number2):
    print(number1 + number2)
```

3.3.20
```
def printTwice(word):
    print(word)
    print(word)
```

3.3.21
```
def printMyName():
    for line in range(100):
        print('Monty Python')
```

Section 3.4

3.4.1
```
"""
Purpose: Draw a flower
Author: Ima Student
Date: September 15, 2020
CS 111
```

```
"""

import turtle

def bloom(tortoise, fcolor, length):
    """Draws a geometric flower bloom.

    Parameters:
        tortoise: a Turtle object with which to draw the bloom.
        fcolor: a color string to use to fill the bloom.
        length: the length of each segment of the bloom.

    Return value:
        None
    """

    tortoise.pencolor('red')     # sets tortoise's pen color to red
    tortoise.fillcolor(fcolor)   # and fill color to fcolor
    tortoise.begin_fill()
    for segment in range(8):     # draw a filled 8-sided geometric
        tortoise.forward(length) #   flower bloom
        tortoise.left(135)
    tortoise.end_fill()

def stem(tortoise, length):
    """Draws a flower stem.

    Parameters:
        tortoise: a Turtle object, initially at the bloom starting
                  position, with which the bloom is drawn.
        length: the length of the stem and each segment of the bloom.

    Return value:
        None
    """

    tortoise.pencolor('green')      # sets tortoise's pen color to green
    tortoise.pensize(length / 20)   # sets pen width wider, proportional to length
    tortoise.up()
    tortoise.forward(length / 2)    # stealthily move to top of stem
    tortoise.down()
    tortoise.right(90)
    tortoise.forward(length)       # draw the stem

def flower(tortoise, fcolor, length):
    """Draws a flower.

    Parameters:
        tortoise: a Turtle object with which to draw the flower.
        fcolor: a color string to use to fill the bloom.
        length: the length of each segment of the bloom.

    Return value:
        None
```

```
    """

    bloom(tortoise, fcolor, length)    # draw the bloom
    stem(tortoise, length)             # and then the stem

def main():
    """Draws a yellow flower with segment length 200, and
       waits for a mouse click to exit."""

    george = turtle.Turtle()
    george.hideturtle()
    george.speed(6)
    flower(george, 'yellow', 200)
    screen = george.getscreen()
    screen.exitonclick()

main()
```

3.4.3

```
import turtle

def drawD(tortoise):
    tortoise.setposition(0, 100)    # move to starting position

    # draw a red filled outer semicircle

    tortoise.pencolor('red')      # set the turtle tail to red
    tortoise.fillcolor('red')     # set the fill color to red
    tortoise.begin_fill()
    tortoise.circle(-100, 180)    # draw 180 degrees of a circle with radius 100
                                  # (negative causes turtle to draw to right)
    tortoise.right(90)            # turn 90 degrees and draw a straight line down
    tortoise.forward(200)
    tortoise.end_fill()

# move to position for inner semicircle

    tortoise.up()
    tortoise.right(90)
    tortoise.forward(25)
    tortoise.right(90)
    tortoise.forward(50)
    tortoise.left(90)
    tortoise.down()

    # draw a white filled inner semicircle

    tortoise.pencolor('white')
    tortoise.fillcolor('white')
    tortoise.begin_fill()
    tortoise.circle(-50, 180)
    tortoise.right(90)
    tortoise.forward(100)
    tortoise.end_fill()

def main():
```

```
    george = turtle.Turtle()    # create a turtle
    drawD(george)
    screen = george.getscreen()
    screen.exitonclick()        # wait for a mouse click to close window

main()
```

Section 3.5

3.5.1
```
def sum(number1, number2):
    return number1 + number2

def main():
    firstNumber = float(input('Number 1: '))
    secondNumber = float(input('Number 2: '))
    print('Sum =', sum(firstNumber, secondNumber))
```

3.5.2
```
def power(base, exponent):
    return base ** exponent

def main():
    base = float(input('Base: '))
    exponent = float(input('Exponent: '))
    print('Answer =', power(base, exponent))
```

3.5.3
```
def football(touchdowns, fieldGoals, safeties):
    score = touchdowns * 7 + fieldGoals * 3 + safeties * 2
    return score

def main():
    td = int(input('Touchdowns: '))
    fg = int(input('Field goals: '))
    safeties = int(input('Safeties: '))
    score = football(td, fg, safeties)
    print('Score =', score)
```

3.5.4
```
def distance(x1, y1, x2, y2):
    return math.sqrt((x1 - x2)**2 + (y1 - y2)**2)
```

3.5.5
```
def moles(V, P, T):
    R = 0.08         # ideal gas constant
    T = T + 273.15   # convert to Kelvin
    return (P * V) / (R * T)

def main():
    volume = float(input('Volume (L): '))
    pressure = float(input('Pressure (atm): '))
    temperature = float(input('Temperature (K): '))
    gas = moles(volume, pressure, temperature)
    print('Moles =', moles(volume, pressure, temperature))
```

3.5.6 The answer to the first part can be computed by calling

```
print(moles(10, 1.5, 20) + moles(25, 2, 30))
```

This computation gives 1.4156011902119108 moles.

If we replace the `return` statement in the `moles` function with a call to `print`, the function will return `None`, so we can no longer perform this computation.

3.5.7
```
def darcy(K, dh, dl):
    return K * dH / dL
```
For the values given, `darcy(130, 50, 1000)` gives 6.5 m^2/day.

3.5.8
```
def bmi(weight, height):
    return 703 * weight / (height * height)
```

3.5.9
```
def songs(capacity, bitrate):
    seconds = 240                                   # 4 minute song in seconds
    capacity_in_bits = capacity * (2 ** 30) * 8     # iPod capacity in bits
    song_space = seconds * bitrate * (2 ** 10)      # bits required for one song

    return capacity_in_bits / song_space            # number of songs
```

3.5.10
```
def time(instructions, gigahertz):
    return instructions * 3 / (gigahertz * 1e9)
```

3.5.11 No, because the parameters a and b are only assigned the values of the arguments x and y. There is no direct connection between a and x, and b and y. If the value assigned to a or b changes, this just changes the value or a or b, not x or y.

3.5.12
```
def gradePoint(score):
    return max(score // 10 - 5, 0)
```

3.5.13
```
def year():
    secYr = 365.25 * 24 * 60 * 60
    return int(time.time() / secYr + 1970)
```

Section 3.6

3.6.1 The modified function is:

```
def distance(x1, y1, x2, y2):
    print('Local namespace:', locals())
    return math.sqrt((x1 - x2)**2 + (y1 - y2)**2)
```

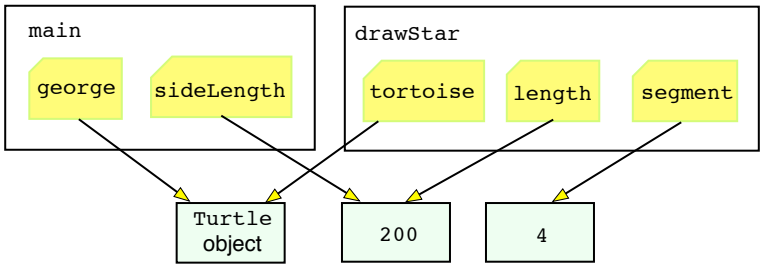
By calling

```
dist = distance(3, 7.25, 9.5, 1)
print(dist)
```

the following should be printed:

```
Local namespace: {'x2': 9.5, 'y1': 7.25, 'y2': 1, 'x1': 3}
9.017344398435716
```

3.6.2



3.6.3 Before the end of `demand`: main: coffee → 5000; demand: quantity → 5. Before the end of `main`: main: coffee → 5000, price → 33.5.

CHAPTER 4

Growth and decay

4.1 EXERCISE SOLUTIONS

Section 4.1

4.1.1
```
for number in range(101):
    print(number)
```

4.1.2
```
def triangle1():
    for count in range(10):
        print('*' * (count + 1))
```

4.1.3
```
def square(letter, width):
    for line in range(width):
        print(letter * width)
```

4.1.4
```
for number in range(-50, 51):
    print(number)
```

4.1.5
```
for number in range(50):
    print(2 * number + 1)
```

4.1.6
```
def clock(minutes):
    for m in range(minutes):
        hours = m // 60
        minutes = m % 60
        print('{0:>2}:{1:0>2}'.format(hours, minutes))
```

4.1.7
```
for number in range(2, 101, 2):
    print(number)

for number in range(1, 100, 2):
    print(number)

for number in range(100, 0, -1):
    print(number)

for number in range(7, 20, 4):
    print(number)

for number in range(2, -3, -1):
    print(number)

for number in range(-7, -20, -4):
    print(number)
```

4.1.8
```
def multiples(n):
    for i in range(0, 101, n):
        print(i)
```

4.1.9
```
def countdown(n):
    for i in range(n, -1, -1):
        print(i)
```

4.1.10
```
def triangle2():
    for count in range(5, 0, -1):
        print('*' * count)
```

4.1.11
```
def triangle3():
    for count in range(6, 0, -1):
        print(' ' * (6 - count) + '*' * count)
```

4.1.12
```
def diamond():
    for count in range(5, 0, -1):
        print('*' * count + ' ' * (11 - 2 * count) + '*' * count)
    for count in range(1, 6):
        print('*' * count + ' ' * (11 - 2 * count) + '*' * count)
```

4.1.13
```
def circles(tortoise):
    for radius in range(10, 110, 10):
        tortoise.up()
        tortoise.right(90)
        tortoise.forward(10)
        tortoise.left(90)
        tortoise.down()
        tortoise.circle(radius)
```

4.1.14
```
tortoise.up()
tortoise.goto(0.5, (2500 - 80 * 0.5) * 0.5 - 8000)
tortoise.down()
```

4.1.15
```
import turtle, math

def plot(tortoise, n):
    for x in range(0, n + 1):
        tortoise.goto(x, math.sin(math.radians(x)))

def main():
    george = turtle.Turtle()
    screen = george.getscreen()
    screen.setworldcoordinates(0, -1, 1080, 1)
    plot(george, 1080)
    screen.exitonclick()

main()
```

4.1.16
```
import turtle, math

def plot(tortoise, n, f):
   tortoise.up()
   tortoise.goto(-n * 10, f(-n))
   tortoise.down()
   for x in range(-n, n + 1):
      tortoise.goto(x * 10, f(x))
```

```
def square(x):
   return x * x

def sin(x):
   return 10 * math.sin(x)

def main():
   george = turtle.Turtle()
   plot(george, 20, sin)
   screen = george.getscreen()
   screen.exitonclick()

main()
```

4.1.17
```
def profitTable(maxPrice):
    print('Price    Income     Profit')
    print('------  ---------  ---------')
    for price in range(1, 4 * maxPrice + 1):
        realPrice = price / 4
        sales = 2500 - 80 * realPrice
        income = sales * realPrice
        profit = income - 8000
        formatString = '${0:>5.2f}  ${1:>8.2f}  ${2:8.2f}'
        print(formatString.format(realPrice, income, profit))
```

4.1.18
```
def pond(years, initialPopulation, harvest, rate):
    population = initialPopulation
    print('Year  Population')
    for year in range(years):
        population = (1 + rate) * population - harvest
        print('{0:^4}  {1:>9.2f}'.format(year + 1, population))

    return population
```

4.1.19
```
def pond(years, initialPopulation, harvest, rate, restock):
    population = initialPopulation
    print('Year  Population')
    for year in range(years):
        population = (1 + rate) * population - harvest + restock
        print('{0:^4}  {1:>9.2f}'.format(year + 1, population))

    return population
```

4.1.20
```
def countLinks(totalNodes):
    totalLinks = 0
    print('|       |   Links     |')
    print('| Nodes | New | Total |')
    print('| ----- | --- | ----- |')
    for node in range(2, totalNodes + 1):
        newLinks = node - 1
        totalLinks = totalLinks + newLinks
        print('| \{0:^5\} | \{1:<3\} | \{2:>5\} |'.format(node, newLinks, totalLinks))

    return totalLinks
```

4.1.21
```
def growth1(totalDays):
    N = 0
    print('Day Size')
```

```
        for day in range(1, totalDays + 1):
            N = N + 3
            print('\{0:<3\}\{1:>5\}'.format(day, N))
        return N
```

4.1.22
```
def growth2(totalDays):
    N = 0
    print('Day Size')
    for day in range(1, totalDays + 1):
        N = N + 2.5
        print('\{0:<3\}\{1:>5\}'.format(day, N))
    return N
```

4.1.23
```
def growth3(totalDays):
    N = 10
    print('Day Size')
    for day in range(1, totalDays + 1):
        N = N * 2.1
        print('\{0:<3\}\{1:>5\}'.format(day, N))
    return N
```

4.1.24
```
def growth4(totalDays):
    N = 10
    print('Day Size')
    for day in range(1, totalDays + 1):
        N = N + 2 ** day
        print('\{0:<3\}\{1:>5\}'.format(day, N))
    return N
```

4.1.25
```
def sum(n):
    total = 0
    for number in range(1, n + 1):
        total = total + number
    return total
```

4.1.26
```
def sumEven(n):
    total = 0
    for number in range(2, n + 1, 2):
        total = total + number
    return total
```

4.1.27
```
def growth(finalAge):
    height = 95
    print('age 3 :', height, 'cm')

    for age in range(4, finalAge + 1):
        height = height + 6
        print('age', age, ':', height, 'cm')

    return height
```

4.1.28
```
def average(low, high):
    total = 0
    for number in range(low, high + 1):
        total = total + number
    return total / (high - low + 1)
```

4.1.29
```
def factorial(n):
    fact = 1
```

```
        for i in range(2, n + 1):
            fact = fact * i
        return fact
```

4.1.30
```
def power(base, exponent):
    answer = 1
    for step in range(exponent):
        answer = answer * base
    return answer
```

4.1.31
```
def geoMean(high):
    product = 1
    for num in range(1, high + 1):
        product = product * num
    return product ** (1 / high)
```

4.1.32
```
def sumDigits(number, digits):
    sum = 0
    for i in range(digits):
        sum = sum + (number % 10)
        number = number // 10
    return sum
```

4.1.33
```
def fun(number, iterations):
    for index in range(iterations):
        digit1 = number // 100
        number = number % 100
        digit2 = number // 10
        digit3 = number % 10
        number = digit1 ** 2 + digit2 ** 2 + digit3 ** 2
        print(number)
```

With every positive integer, either the sequence reaches (and stays at) 1 or it reaches 4 and forever repeats the sequence $4, 16, 37, 58, 89, 145, 42, 20, 4, \ldots$. The numbers that cause the sequence to reach 1 are known as the "happy numbers." The first few happy numbers are $1, 7, 10, 13, 19, 23, 28, \ldots$.

4.1.34
```
def interest(originalAmount, rate, periods):
    amount = originalAmount
    rate = rate / 100 / periods
    for p in range(periods):
        amount = amount + rate * amount
    return amount - originalAmount

print(interest(1000, 1.0, 365))
print(interest(1000, 1.25, 12))
```

4.1.35
```
def invest(investment, rate, years):
    amount = 0
    rate = rate / 100 / 12
    for p in range(years * 12):
        amount = amount + rate * amount + investment
    return amount - (investment * years * 12)

print(invest(50, 1.0, 10))
```

4.1.36
```
def mortgage(principal, rate, years, payment):
    amount = principal
    interest = 0
    print('Month   Payment     Balance')
```

```
        for step in range(years * 12 - 1):
            newInterest = rate * amount / 12
            interest = interest + newInterest
            amount = amount + newInterest - payment

            print('{0:<5}  {1:>8.2f}  {2:>10.2f}'.format(step + 1, payment, amount))

        newInterest = rate * amount / 12
        interest = interest + newInterest
        amount = amount + newInterest
        print('{0:<5}  {1:>8.2f}  {2:>10.2f}'.format(years * 12, amount, 0))

    mortgage(200000, 0.05, 30, 1073.64)
```

4.1.37
```
    def bacteria(days):
        population = 100
        hours = days * 24
        for hour in range(hours):
            population = population + 0.1 * population
        return population
```

4.1.38
```
    def bacteria(days, population, rate):
        hours = days * 24
        for hour in range(hours):
            population = population + rate * population
        return population
```

Section 4.2

4.2.1
```
    import matplotlib.pyplot as pyplot

    def countLinks(totalNodes):
        totalLinks = 0
        linksList = [0]
        for node in range(2, totalNodes + 1):
            newLinks = node - 1
            totalLinks = totalLinks + newLinks
            linksList.append(totalLinks)

        pyplot.plot(list(range(1, totalNodes + 1)), linksList)
        pyplot.xlabel('Nodes')
        pyplot.ylabel('Maximum number of links')
        pyplot.show()

        return totalLinks
```

4.2.2
```
    def profitPlot(maxPrice):
        profitList = []
        priceList = []
        for price in range(1, 2 * maxPrice + 1):
            realPrice = price / 2
            sales = 2500 - 80 * realPrice
            income = sales * realPrice
            profit = income - 8000
            priceList.append(realPrice)
            profitList.append(profit)
```

```
        pyplot.plot(priceList, profitList)
        pyplot.xlabel('Ticket price ($)')
        pyplot.ylabel('Profit ($)')
        pyplot.show()
```

4.2.3
```
def growth1(totalDays):
    N = 0
    popList = [0]
    for day in range(1, totalDays + 1):
        N = N + 3
        popList.append(N)
    pyplot.plot(range(totalDays + 1), popList)
    pyplot.xlabel('Day')
    pyplot.ylabel('Population size')
    pyplot.show()
    return N
```

4.2.4
```
def growth3(totalDays):
    N = 10
    popList = [10]
    for day in range(1, totalDays + 1):
        N = N * 1.1
        popList.append(N)
    pyplot.plot(range(totalDays + 1), popList)
    pyplot.xlabel('Day')
    pyplot.ylabel('Population size')
    pyplot.show()
    return N
```

4.2.5
```
def invest(investment, rate, years):
    amount = 0
    amountList = []
    rate = rate / 100 / 12
    for p in range(years * 12):
        amount = amount + rate * amount + investment
        amountList.append(amount)

    pyplot.plot(range(years * 12), amountList)
    pyplot.xlabel('Month')
    pyplot.ylabel('Amount (dollars)')
    pyplot.show()

    return amount - (investment * years * 12)

print(invest(50, 8.0, 10))
```

Section 4.3

4.3.1
```
amount = 1000
year = 0
while amount < 1200:
    amount = 1.03 * amount
    year = year + 1
print(year)
```

4.3.2
```
def interest(amount, rate, target):
```

```
    years = 0
    while amount < target:
        amount = (1 + rate) * amount
        years = years + 1
    return years
```

4.3.3 (a) If we omit the initialization of `counter`, we get error at the `while` loop saying that the variable was referenced before it was assigned. This means that the interpreter does not know what `counter` is the first time the condition is tested because it has not yet been defined. We have to always make sure the condition makes sense before the `while` loop starts.

(b) The loop becomes an infinite loop, printing 0 forever. We have to always make sure in the body of the `while` loop to work toward the condition eventually become false.

(c)
```
index = 3
while index < 12:
    print(index)
    index = index + 1
```

(d)
```
index = 12
while index > 3:
    print(index)
    index = index - 1
```

4.3.4
```
def profitTable(maxPrice):
    print('Price    Income     Profit')
    print('------  ---------  ---------')
    price = 1.0
    while price <= maxPrice:
        sales = 2500 - 80 * price
        income = sales * price
        profit = income - 8000
        formatString = '${0:>5.2f}  ${1:>8.2f}  ${2:8.2f}'
        print(formatString.format(price, income, profit))
        price = price + 0.5
```

4.3.5
```
def zombieApocalypse():
    zombies = 1
    days = 0
    while zombies < 7e9:
        newZombies = zombies * 2
        zombies = zombies + newZombies
        days = days + 1
    return days
```

4.3.6
```
def tribbleApocalypse():
    tribbles = 10
    hours = 0
    while tribbles < 1e6:
        tribbles = tribbles + tribbles // 2
        hours = hours + 1
    return hours
```

4.3.7
```
def vampireApocalypse(v, k, vampires, people):
    days = 0
    while people > 0:
        people = people - vampires * v
        vampires = vampires + vampires * v - k
```

```
            days = days + 1
        return days
```

4.3.8
```
def amoebaGrowth(h, target):
    hours = 0
    amoebas = 1
    while amoebas < target:
        amoebas = 2 * amoebas
        hours = hours + h
    return hours
```

Section 4.4

4.4.1 The difference equation is

$$R(m) = R(m-1) + R(m-2)$$

where $R(m-1)$ represents the number of rabbits alive the previous month and $R(m-2)$ represents a new pair for every pair alive two months ago. The initial conditions are $R(1) = R(2) = 1$. This famous sequence is known as the Fibonacci numbers.

```
import matplotlib.pyplot as pyplot

def rabbits(months):
    population2 = 0   # R(m-2)
    population1 = 1   # R(m-1)
    populationList = [1]
    for month in range(1, months + 1):
        population = population1 + population2  # R(m)
        population2 = population1
        population1 = population
        populationList.append(population)
    pyplot.plot(range(months + 1), populationList)
    pyplot.xlabel('Month')
    pyplot.ylabel('Rabbit Population Size')
    pyplot.show()
    return population
```

4.4.2 The difference equation is

$$B(t) = B(t-\Delta t) + 0.1 * B(t-\Delta t) * \Delta t$$

when $t > 0$. The initial condition is $B(0) = 100$.

```
def bacteria(population, dt, days):
    numIterations = int(24 * days / dt) + 1
    for t in range(numIterations):
        population = population + 0.1 * population * dt
    return population
```

4.4.3
```
def halflifeC14(originalAmount, dt):
    amount = originalAmount
    k = -0.00012096809434
    iterations = 0
    while amount > originalAmount / 2:
        amount = amount + k * amount * dt
        iterations = iterations + 1
    return iterations * dt
```

4.4.4
```
def carbonDate(originalAmount, fractionRemaining, dt):
    amount = originalAmount
    k = -0.00012096809434
    iterations = 0
    while amount > originalAmount * fractionRemaining:
        amount = amount + k * amount * dt
        iterations = iterations + 1
    return iterations * dt
```
The parchment is approximately 2948.5 years old.

4.4.5
```
    numIterations = int(days/dt) + 1
    for i in range(1, numIterations):
        t = i * dt
        dR = (recRate * infected) * dt                # newly recovered
        dS = (infRate * susceptible * infected) * dt  # newly infected

        recovered = recovered + dR
        susceptible = susceptible - dS
        infected = infected + dS - dR

        timeList.append(t)
        SList.append(susceptible)
        IList.append(infected)
        RList.append(recovered)
```

4.4.6 No, about 73 people remain healthy.

4.4.7 The peak number of infecteds is cut about in half, and about 620 people remain healthy.

4.4.8
$$S(t) = S(t-\Delta t) - dS(t-\Delta t)I(t-\Delta t)\Delta t + rI(t-\Delta t)\Delta t$$
$$I(t) = I(t-\Delta t) + dS(t-\Delta t)I(t-\Delta t)\Delta t - rI(t-\Delta t)\Delta t$$

```
import matplotlib.pyplot as pyplot

def SIS(population, dt, days):
    numIterations = int(days/dt) + 1
    susceptible = population - 1
    infected = 1.0
    recRate = 0.25
    infRate = 0.0004
    SList = [susceptible]
    IList = [infected]
    timeList = [0]

    for i in range(1, numIterations):
        t = i * dt
        dR = (recRate * infected) * dt                # newly recovered
        dS = (infRate * susceptible * infected) * dt  # newly infected

        susceptible = susceptible - dS + dR
        infected = infected + dS - dR

        timeList.append(t)
        SList.append(susceptible)
        IList.append(infected)
```

```
    pyplot.plot(timeList, SList, label = 'Susceptible')
    pyplot.plot(timeList, IList, label = 'Infected')
    pyplot.legend(loc = 'center right')
    pyplot.xlabel('Days')
    pyplot.ylabel('Individuals')
    pyplot.show()
```

The populations reach an equilibrium.

4.4.9
```
import matplotlib.pyplot as pyplot

def compete(pop1, pop2, birth1, birth2, death1, death2, years, dt):
    pop1List = [pop1]
    pop2List = [pop2]
    timeList = [0]

    for step in range(1, int(years / dt) + 1):
        t = step * dt
        newPop1 = (birth1 * pop1 - death1 * pop1 * pop2) * dt
        newPop2 = (birth2 * pop2 - death2 * pop2 * pop1) * dt
        pop1 = pop1 + newPop1
        pop2 = pop2 + newPop2

        timeList.append(t)
        pop1List.append(pop1)
        pop2List.append(pop2)

    pyplot.plot(timeList, pop1List, label = 'Population 1')
    pyplot.plot(timeList, pop2List, label = 'Population 2')
    pyplot.legend()
    pyplot.xlabel('Years')
    pyplot.ylabel('Individuals')
    pyplot.show()

def main():
    compete(21, 26, 1.0, 1.02, 0.2, 0.25, 6, 0.001)

main()
```

Section 4.5

4.5.1 (a)
```
def ant():
    total = 0
    step = 0
    while total < 10:
        step = step + 1
        total = total + (1 / step)
    return step
```

(b) 12,367 minutes = over 8.5 days

(c) about 272,459,350 minutes = over 518 years

4.5.2
```
import matplotlib.pyplot as pyplot, math

def ant(n):
    total = 0
```

```
    y = [ ]
    y2 = [ ]
    for number in range(1, n + 1):
        total = total + (1 / number)
        y.append(total)
        y2.append(math.log(number) + 0.577)
    pyplot.plot(range(1, n + 1), y, label = 'Harmonic series')
    pyplot.plot(range(1, n + 1), y2, label = 'ln n + 0.577')
    pyplot.xlabel('n')
    pyplot.ylabel('y')
    pyplot.legend(loc = 'center right')
    pyplot.show()
    return total
```

4.5.3
```
def e(n):
    sum = 1
    for i in range(1, n):
        sum = sum + 1.0 / factorial(i)
    return sum
```

4.5.4 The value of `factorial(n)` is equal to `factorial(n-1) * n`. Therefore, every time `factorial(n)` is called, all of the previous factorials are being computed again.

4.5.5
```
def e(n):
    sum = 1
    factorial = 1
    for i in range(1, n):
        factorial = factorial * i
        sum = sum + 1.0 / factorial
    return sum
```

4.5.6
```
def leibniz():
    sum = 0
    term = 1
    index = 0
    while abs(term) > 1e-6:
        term = (-1) ** index / (2 * index + 1)
        sum = sum + term
        index = index + 1
    pi = sum * 4
    return pi
```

4.5.7
```
def sqrt(n):
    x = 1.0
    prevX = 0
    while abs(x - prevX) > 1e-15:
        prevX = x
        x = 0.5 * (x + n / x)
    return x
```

4.5.8
```
def approxPi(n):
    sum = 0
    for i in range(terms):
        sum = sum + (-1) ** i / (3 ** i * (2 * i + 1))
    sum = sum * math.sqrt(12)
    return sum
```

4.5.9
```
def wallis(terms):
    pairs = terms / 2
```

```
    product = 1
    for i in range(pairs):
        first = (2.0 * (i + 1)) / (2 * i + 1)
        second = (2.0 * (i + 1)) / (2 * i + 3)
        product = product * first * second
    pi = product * 2
    return pi
```

4.5.10
```
def nilakantha(terms):
    sum = 3
    for i in range(1, terms + 1):
        sum = sum + (-1)**(i-1) * 4 / ((2 * i) * (2 * i + 1) * (2 * i + 2))
    pi = sum
    return pi
```

4.5.11
```
def vieta(terms):
    product = 2
    term = 0
    for i in range(1, terms + 1):
        term = math.sqrt(2 + term)
        product = product *  2 / term
    pi = product
    return pi
```

Section 4.6

4.6.1 (a) The accumulator exhibits quadratic growth.

(b) The accumulator exhibits quadratic growth.

(c) The accumulator exhibits exponential growth.

(d) The accumulator exhibits *no* growth. Since `sum` is initialized to zero, its final value is also zero!

(e) The accumulator exhibits linear growth.

(f) The accumulator exhibits exponential growth.

4.6.2 Because the exponential curve is generated by multiplying the accumulator by a number close to 1, it initially grows slowly. However, once the accumulator reaches larger values, the multiplication causes it to increase more rapidly.

4.6.3
```
import matplotlib.pyplot as pyplot

def accumulator(n):
    sum1 = 0
    sum2 = 0
    sum3 = 1
    list1 = []
    list2 = []
    list3 = []
    for index in range(n):
        sum1 = sum1 + 6
        sum2 = sum2 + index
        sum3 = sum3 + 0.08 * sum3
        list1.append(sum1)
        list2.append(sum2)
        list3.append(sum3)
```

```
        pyplot.plot(range(n), list1, label = 'linear', color = 'blue')
        pyplot.plot(range(n), list2, label = 'quadratic', color = 'red')
        pyplot.plot(range(n), list3, label = 'exponential', color = 'darkgreen')
        pyplot.legend(loc = 'upper center')
        pyplot.xlabel('n')
        pyplot.ylabel('sum')
        pyplot.xlim(0, 125)
        pyplot.show()

    accumulator(125)
```

4.2 PROJECT SOLUTIONS

Project 4.1

```
import math
import matplotlib.pyplot as pyplot

def NB(hostPop, paraPop, r, c, a, years):
'''Simulate the Nicholson-Bailey model.'''

hostList = [hostPop]
paraList = [paraPop]

for year in range(years):
escapeProb = math.exp(-a * paraPop)
newHostPop = r * hostPop * escapeProb
newParaPop = c * hostPop * (1 - escapeProb)

hostPop = newHostPop
paraPop = newParaPop
hostList.append(hostPop)
paraList.append(paraPop)

pyplot.plot(hostList, paraList)
pyplot.xlabel('Hosts')
pyplot.ylabel('Parasitoids')
pyplot.show()

pyplot.plot(range(years + 1), paraList, label = 'Parasitoids')
pyplot.plot(range(years + 1), hostList, label = 'Hosts')
pyplot.legend()
pyplot.xlabel('Years')
pyplot.ylabel('Individuals')
pyplot.show()

def NB_CC(hostPop, paraPop, r, c, a, K, years):
'''Simulate the Nicholson-Bailey model with carrying capacity.'''

hostList = [hostPop]
paraList = [paraPop]

for year in range(years):
escapeProb = math.exp(-a * paraPop)
```

```
newHostPop = hostPop * escapeProb * math.exp(r * (1 - hostPop / K))
newParaPop = c * hostPop * (1 - escapeProb)

hostPop = newHostPop
paraPop = newParaPop

hostList.append(hostPop)
paraList.append(paraPop)

pyplot.plot(hostList, paraList)
pyplot.xlabel('Hosts')
pyplot.ylabel('Parasitoids')
pyplot.show()

pyplot.plot(range(years + 1), paraList, label = 'Parasitoids')
pyplot.plot(range(years + 1), hostList, label = 'Hosts')
pyplot.legend()
pyplot.xlabel('Years')
pyplot.ylabel('Individuals')
pyplot.show()

def main():
    hosts = 24
    parasitoids = 12
    r = 2
    c = 1
    a = 0.056 # -math.log(0.5) / paraPop
    years = 35

    NB(hosts, parasitoids, r, c, a, years)

    r = 1.5
    K = 40

    NB_CC(hosts, parasitoids, r, c, a, K, years)

main()
```

Project 4.3

```
import matplotlib.pyplot as pyplot

def productDiffusion(adoptRate, socialContagion, weeks, dt):
    adopters = 0
    adoptersList = [adopters]
    newAdoptersList = [adopters]
    timeList = [0]

    for i in range(1, int(weeks / dt) + 1):
        newAdopters = (adoptRate * (1 - adopters) + socialContagion * adopters
                       * (1 - adopters)) * dt
        adopters = adopters + newAdopters

        adoptersList.append(adopters)
```

```
        newAdoptersList.append(newAdopters / dt)  # weekly rate
        timeList.append(i * dt)

    pyplot.plot(timeList, newAdoptersList, label = 'New adopters')
    pyplot.ylabel('Fraction of adopters')
    pyplot.plot(timeList, adoptersList, label = 'Total adopters')
    pyplot.xlabel('Week')
    pyplot.legend()
    pyplot.show()

def productDiffusion2(inSize, imSize, rIn, sIn, rIm, sIm, weight, weeks, dt):
    influentials = 0
    imitators = 0
    influentialsList = [influentials]
    imitatorsList = [imitators]
    adoptersList = [influentials + imitators]
    newInfluentialsList = [influentials]
    newImitatorsList = [imitators]
    newAdoptersList = [influentials + imitators]
    timeList = [0]

    for i in range(1, int(weeks / dt) + 1):
        newInfluentials = (rIn * (1 - influentials) + sIn * influentials
                           * (1 - influentials)) * dt
        newImitators = (rIm * (1 - imitators) + sIm * (weight * influentials
                       * (1 - imitators) + (1 - weight) * imitators
                       * (1 - imitators))) * dt

        influentials = influentials + newInfluentials
        imitators = imitators + newImitators

        influentialsList.append(influentials * inSize)
        imitatorsList.append(imitators * imSize)
        adoptersList.append(influentials * inSize + imitators * imSize)
        newInfluentialsList.append(newInfluentials * inSize / dt)  # weekly rates
        newImitatorsList.append(newImitators * imSize / dt)
        newAdoptersList.append(newInfluentials * inSize / dt + newImitators * imSize / dt)
        timeList.append(i * dt)

    pyplot.plot(timeList, newInfluentialsList, label = 'Influentials')
    pyplot.plot(timeList, newImitatorsList, label = 'Imitators')
    pyplot.plot(timeList, newAdoptersList, label = 'Total', color = 'black')
    pyplot.xlabel('Week')
    pyplot.ylabel('New adoptions')
    pyplot.legend()
    pyplot.show()

    pyplot.plot(timeList, influentialsList, label = 'Influentials')
    pyplot.plot(timeList, imitatorsList, label = 'Imitators')
    pyplot.plot(timeList, adoptersList, label = 'Total', color = 'black')
    pyplot.xlabel('Week')
    pyplot.ylabel('Total adoptions')
    pyplot.legend(loc = 'upper left')
    pyplot.show()
```

```
def main():
    productDiffusion(0.002, 1.03, 15, 0.01)
    productDiffusion2(600, 400, 0.002, 1.03, 0, 0.8, 0.6, 15, 0.01)
```

Project 4.4

```
import matplotlib.pyplot as pyplot

def PP(preyPop = 100.0, predPop = 15.0, dt = 0.01, months = 12):
    '''Simulate predator-prey model.'''

    numIterations = int(months/dt) + 1

    preyBirthRate = 0.5
    preyDeathRate = 0.02  # rate at which predator kills prey it encounters
    efficiency = 0.25     # efficiency of wolf turning an eaten sheep into a new wolf
    predBirthRate = preyDeathRate * efficiency
    predDeathRate = 0.75

    preyList = [preyPop]
    predList = [predPop]
    t = 0.0
    timeList = [0]

    for i in range(1, numIterations):
        t = i * dt                      # minimize accumulation error
        timeList.append(t)

        dPred = (predBirthRate * preyPop * predPop - predDeathRate * predPop) * dt
#       dPrey = (preyBirthRate * preyPop - preyDeathRate * preyPop * predPop) * dt
        dPrey = (preyBirthRate * (1.0 - preyPop / 750.0) * preyPop
                - preyDeathRate * preyPop * predPop) * dt

        preyPop += dPrey
        predPop += dPred

        if preyPop < 1.0:
            preyPop = 0.0
        if predPop < 1.0:
            predPop = 0.0

        preyList.append(preyPop)
        predList.append(predPop)

        print preyPop, predPop

    #pyplot.plot(preyList, predList)
    pyplot.plot(timeList, predList, label = 'Predators')
    pyplot.plot(timeList, preyList, label = 'Prey')
    pyplot.legend(loc = 'upper right')
    pyplot.xlabel('Months')
    pyplot.ylabel('Individuals')
    pyplot.show()
```

CHAPTER 5

Forks in the road

5.1 EXERCISE SOLUTIONS

Section 5.1

5.1.1
```
if r < 0.25:    # if r < 0.25,
    x = x + 1   #    move to the east and finish
elif r < 0.4:   # otherwise, if r is in [0.25, 0.4),
    y = y + 1   #    move to the north and finish
elif r < 0.65:  # otherwise, if r is in [0.4, 0.65),
    x = x - 1   #    move to the west and finish
else:           # otherwise,
    y = y - 1   #    move to the south and finish
```

5.1.2
```
def weather():
    r = random.random()
    if r < 0.7:
        print('RAIN!')
    else:
        print('SUN!')
```

5.1.3
```
def loaded():
    r = random.random()
    if r < 0.25:
        roll = 1
    elif r < 0.5:
        roll = 6
    elif r < 0.625:
        roll = 2
    elif r < 0.75:
        roll = 3
    elif r < 0.875:
        roll = 4
    else:
        roll = 5
    return roll
```

5.1.4
```
def roll():
    r = random.random()
    if r < 1 / 6:
        return 1
    elif r < 1 / 3:
        return 2
```

```
        elif r < 1 / 2:
            return 3
        elif r < 2 / 3:
            return 4
        elif r < 5 / 6:
            return 5
        else:
            return 6

    def diceHistogram(trials):
        rolls = []
        for trial in range(trials):
            sum = roll() + roll()
            rolls.append(sum)
        pyplot.hist(rolls, 11)
        pyplot.show()
```

5.1.5
```
    def montyHall(choice, switch):
        car = 2
        if choice == car:
            if switch:
                win = False
            else:
                win = True
        else:
            if switch:
                win = True
            else:
                win = False
        return win

    def monteMonty(trials):
        wins = 0
        for trial in range(trials):
            choice = int(random.random() * 3)
            if montyHall(choice, True):
                wins = wins + 1
        return wins / trials
```

The Monte Carlo simulation shows that switching doors gives a contestant a 2/3 chance of winning the car.

This is because the probability of winning the car if you switch doors is exactly the probability of initially choosing a door with a goat, which is 2/3. To see why, consider two possibilities. First, if the contestant initially chooses the door with the car, but switches, he does not win the car. On the other hand, suppose the contestant initially chooses a door with a goat. After Monty reveals the location of the other goat, there are two doors left: the one that the contestant chose (with a goat) and the one that must hide the car. Therefore, switching means the contestant wins.

5.1.6
```
    def montePi(darts):
        circle = 0                              # init number of points in circle
        for trials in range(darts):             # iterate over the number of darts
            x = random.random()                 # get a random x coordinate
            y = random.random()                 # get a random y coordinate
            distance = math.sqrt(x**2 + y**2)   # compute distance from (0,0)
            if distance <= 1:                   # if (x,y) is in the circle then
```

```
                circle = circle + 1            #   count it
        pi = (circle / darts) * 4              # area = pi * (1)^2
        return pi
```

5.1.7

```
def goodBadUgly():
    good = good1 = 1
    bad = bad1 = 1
    ugly = ugly1 = 1
    rounds = 0
    while good + bad + ugly > 1:
        rounds = rounds + 1
        if good == 1:
            if random.random() < 0.8:
                if bad == 1:
                    bad1 = 0
                else:
                    ugly1 = 0
        if bad == 1:
            if random.random() < 0.7:
                if good == 1:
                    good1 = 0
                else:
                    ugly1 = 0
        if ugly == 1:
            if random.random() < 0.6:
                if bad == 1:
                    bad1 = 0
                else:
                    good1 = 0
        good = good1
        bad = bad1
        ugly = ugly1
    if good == 1:
        return 1, rounds
    if bad == 1:
        return 2, rounds
    if ugly == 1:
        return 3, rounds
    return 0, rounds

def monteGBU(trials):
    totalGood = 0
    totalBad = 0
    totalUgly = 0
    totalNone = 0
    totalRounds = 0
    for trial in range(trials):
        winner, rounds = goodBadUgly()
        if winner == 1:
            totalGood = totalGood + 1
        elif winner == 2:
            totalBad = totalBad + 1
        elif winner == 3:
            totalUgly = totalUgly + 1
        else:
```

```
                totalNone = totalNone + 1
            totalRounds = totalRounds + rounds
        print('Probability that everyone dies =', totalNone / trials)
        print('Probability that The Good survives =', totalGood / trials)
        print('Probability that The Bad survives =', totalBad / trials)
        print('Probability that The Ugly survives =', totalUgly / trials)
        print('Average number of rounds =', totalRounds / trials)
```

5.1.8 (a) `Candidate two wins!`

(b) `Candidate one wins!`

(c) `Candidate one wins!`

5.1.9
```
if votes1 > votes2:
    print('Candidate one wins!')
elif votes1 < votes2:
    print('Candidate two wins!')
else:
    print('There was a tie.')
```

5.1.10 (a) `A runoff is required.`

(b) `Candidate one wins!` and `A runoff is required.`

(c) `Candidate two wins!` and `A runoff is required.`

5.1.11
```
majority = (votes1 + votes2 + votes3) / 2
if votes1 > majority:
    print('Candidate one wins!')
elif votes2 > majority:
    print('Candidate two wins!')
elif votes3 > majority:
    print('Candidate three wins!')
else:
    print('A runoff is required.')
```

5.1.12 The `elif` has no associated `if`.

5.1.13 (a) 27

(b) 25

5.1.14
```
def computeGrades(scores):
    grades = []
    for score in scores:
        if score >= 90:
            grades.append('A')
        elif score >= 80:
            grades.append('B')
        elif score >= 70:
            grades.append('C')
        elif score >= 60:
            grades.append('D')
        else:
            grades.append('F')
    return grades
```

Section 5.2

5.2.1
```
import random, turtle
```

```
def testRandom(n):
    tortoise = turtle.Turtle()
    screen = tortoise.getscreen()
    screen.setworldcoordinates(0, 0, 1, 1)
    screen.tracer(100)
    tortoise.up()
    tortoise.speed(0)

    for point in range(n):
        x = random.random()
        y = random.random()

        tortoise.goto(x, y)
        tortoise.dot()

    screen.update()
    screen.exitonclick()
```

5.2.2

```
import random, turtle

def lehmer(r, m, a):
    return (a * r) % m

def testRandom(n):
    tortoise = turtle.Turtle()
    screen = tortoise.getscreen()
    screen.setworldcoordinates(0, 0, 1, 1)
    screen.tracer(100)
    tortoise.up()
    tortoise.speed(0)

    r = 10
    m = 2**31 - 1
    a = 16807
    for index in range(n):
        r1 = lehmer(r, m, a)
        r2 = lehmer(r1, m, a)
        x = r1 / m
        y = r2 / m
        tortoise.goto(x, y)
        tortoise.dot()
        r = r2

    screen.update()
    screen.exitonclick()
```

5.2.3

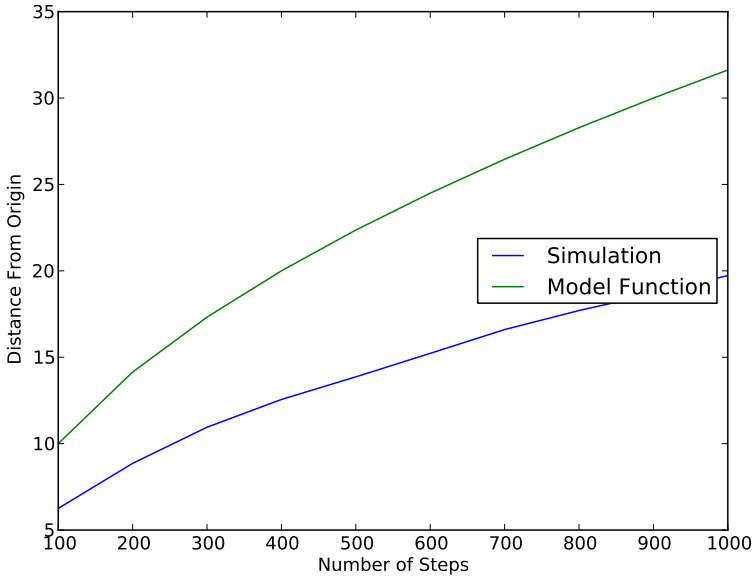
```
import random

def avgTest(n):
    sum = 0
    for trial in range(n):
        sum = sum + random.random()

    return sum / n
```

Section 5.3

5.3.1
```
def randomWalk(steps, tortoise, draw):
    x = 0
    y = 0
    moveLength = 10
    for step in range(steps):
        x = x + random.gauss(0, 0.5)
        y = y + random.gauss(0, 0.5)
        if draw:
            tortoise.goto(x * moveLength, y * moveLength)
    return distance(0, 0, x, y)
```



5.3.2
```
def uniform(a, b):
    return a + random.random() * (b - a)
```

5.3.3
```
number = int(random.random() * 8)
```

5.3.4
```
def randomRange(a, b):
    return int(a + random.random() * (b - a + 1))
```

5.3.5
```
def normalHist(mean, stdDev, trials):
    samples = [ ]
    for i in range(trials):
        samples.append(random.gauss(mean, stdDev))
    pyplot.hist(samples, 100)
    pyplot.show()
```

5.3.6
```
def uniformHist(a, b, trials):
    samples = [ ]
    for i in range(trials):
        samples.append(random.uniform(a, b))
    pyplot.hist(samples, 100)
    pyplot.show()
```

5.3.7
```
def sumGaussSquared(k):
```

```
        sum = 0
        for j in range(k):
            sum = sum + random.gauss(0, 1) ** 2
        return sum

    def plotChiSquared(k, trials):
        samples = [ ]
        for i in range(trials):
            sum = sumGaussSquared(k)
            samples.append(sum)
        pyplot.hist(samples, 200)
        pyplot.show()
```

Section 5.4

5.4.1
```
def password():
    username = input('Username: ')
    password = input('Password: ')
    return username == 'alan.turing' and password == 'notTouring'
```

5.4.2
```
def hasWon(score):
    return score >= 100
```

5.4.3
```
def monitor(temperature):
  return temperature >= 97.9 and temperature <= 99.3
```

5.4.4
```
def winner(score1, score2):
    if score1 > score2:
        return 1
    else:
        return 2
```

5.4.5
```
def winner(score1, score2):
    if score1 > score2:
        return 1
    elif score2 > score1:
        return 2
    else:
        return 0
```

5.4.6
```
def cost(quantity):
    if quantity <= 20:
        return quantity * 1500
    else:
        return quantity * 1500 * 0.95
```

5.4.7
```
def cost(quantity, cost, discountQuantity, discount):
    if quantity <= discountQuantity:
        return quantity * cost
    else:
        return quantity * cost * (1 - discount)
```

5.4.8
```
def fine(speedLimit, clockedSpeed):
    if clockedSpeed <= speedLimit:
        return 0
    amount = 50 + 5 * (clockedSpeed - speedLimit)
    if clockedSpeed > 90:
        amount = amount + 200
    return amount
```

5.4.9
```
def gradeRemark():
    grade = float(input('Grade: '))
    if grade >= 96:
        return 'Outstanding'
    elif grade >= 90:
        return 'Exceeds expectations'
    elif grade >= 80:
        return 'Acceptable'
    else:
        return 'Trollish!'
```

5.4.10
```
def f(a, b):
   if a != b:
     return a + b
   else:
     return a * b
```

5.4.11
```
def amIRich(amount, rate, years):
   finalAmount = amount
   for i in range(years):
     finalAmount = finalAmount * (1 + rate)
   return finalAmount >= 2 * amount
```

5.4.12
```
def max3(a, b, c):
    if (a >= b) and (a >= c):
        return a
    elif b >= c:
        return b
    else:
        return c
```

5.4.13
```
def shipping(amount):
    if amount <= 100:
        return 6.95
    return 6.95 + 0.05 * (amount - 100)
```

5.4.14
```
def mystery(a, b):
    sum = (a + b) % 10
    first = b
    second = sum
    count = 3
    while first != a or second != b:
        sum = (first + second) % 10
        first = second
        second = sum
        count = count + 1
    return count
```

5.4.15
```
def zodiac(year):
    index = year % 12
    if index == 0:
        return 'Monkey'
    if index == 1:
        return 'Rooster'
    if index == 2:
        return 'Dog'
    if index == 3:
        return 'Pig'
```

```
    if index == 4:
        return 'Rat'
    if index == 5:
        return 'Ox'
    if index == 6:
        return 'Tiger'
    if index == 7:
        return 'Rabbit'
    if index == 8:
        return 'Dragon'
    if index == 9:
        return 'Snake'
    if index == 10:
        return 'Horse'
    if index == 11:
        return 'Goat'
```

5.4.16
```
def leap(year):
    if year % 100 == 0:
        return year % 400 == 0
    return year % 4 == 0
```

5.4.17
```
def even(number):
    return (number % 2 == 0)
```

5.4.18
```
def between(number, low, high):
    return (number >= low and number <= high)
```

5.4.19
```
def justone(a, b):
    return (a == 10 and b != 10) or (a != 10 and b == 10)
```

5.4.20
```
def roll():
    sum = loaded() + loaded()
    return sum == 7 or sum == 11
```

5.4.21
```
def perfectSquare(number):
    return (number >= 0) and (math.sqrt(number) == int(math.sqrt(number)))
```

5.4.22
```
def rwMonteCarlo(steps, trials):
    if trials <= 0:
        return 0
    totalDistance = 0
    for trial in range(trials):
        distance = randomWalk(steps, None, False)
        totalDistance = totalDistance + distance
    return totalDistance / trials
```

5.4.23
```
(x >= -d) and (x <= d) and (y >= -d) and (y <= d)
```

5.4.24

x > d	x < -d	y > d	x > d and y > d	x < -d and y > d	(x > d and y > d) or (x < -d and y > d)
F	F	F	F	F	F
F	F	T	F	F	F
F	T	F	F	F	F
F	T	T	F	T	T
T	F	F	F	F	F
T	F	T	T	F	T
T	T	F	F	F	F
T	T	T	T	T	T

x > d	x < -d	y > d	x > d or x < -d	(y > d) and (x > d or x < -d)
F	F	F	F	F
F	F	T	F	F
F	T	F	T	F
F	T	T	T	T
T	F	F	T	F
T	F	T	T	T
T	T	F	T	F
T	T	T	T	T

Since the last columns of the two truth tables are the same, the expressions are equivalent.

5.4.25
```
def drawRow(tortoise, row):
    for column in range(8):
        if (row + column) % 2 == 0:
            tortoise.fillcolor('red')
        else:
            tortoise.fillcolor('black')
        tortoise.begin_fill()
        drawSquare(tortoise, 100)
        tortoise.end_fill()
        tortoise.forward(100)
```

5.4.26
```
def checkerBoard(tortoise):
    tortoise.up()
    tortoise.backward(500)
    tortoise.right(90)
    tortoise.backward(500)
    tortoise.left(90)
    tortoise.down()

    for row in range(8):
        drawRow(tortoise, row)
        tortoise.up()
        tortoise.backward(800)
        tortoise.right(90)
        tortoise.forward(100)
        tortoise.left(90)
        tortoise.down()
```

Section 5.5

5.5.1
```
def ABC():
    answer = input('Enter A, B, or C: ')
    while answer != 'A' and answer != 'B' and answer != 'C':
        answer = input('Enter A, B, or C: ')
    print('Thank you.')
```

5.5.2
```
def numberPlease():
    answer = int(input('Enter a number between 1 and 100 (inclusive): '))
    while answer < 1 or answer > 100:
        answer = int(input('Enter a number between 1 and 100 (inclusive): '))
    print('Thank you.')
```

5.5.3
```
def differentNumbers():
    number1 = float(input('Enter one number: '))
```

```
        number2 = float(input('Enter a different number: '))
        while number1 == number2:
            number1 = float(input('Enter one number: '))
            number2 = float(input('Enter a different number: '))
        print('Thank you.')
```

5.5.4
```
def rockPaperScissorsLizardSpock(player1, player2):
    '''Return 1 if player1 wins, -1 if player2 wins, or 0 if they tie.'''

    if player1 == player2:
        result = 0
    if player1 == 'scissors' and (player2 == 'paper' or player2 == 'lizard'):
        result = 1
    elif player1 == 'paper' and (player2 == 'rock' or player2 == 'Spock'):
        result = 1
    elif player1 == 'rock' and (player2 == 'lizard' or player2 == 'scissors'):
        result = 1
    elif player1 == 'lizard' and (player2 == 'Spock' or player2 == 'paper'):
        result = 1
    elif player1 == 'spock' and (player2 == 'scissors' or player2 == 'rock'):
        result = 1
    else:
        result = -1

    return result
```

5.5.5
```
def yearsUntilDoubled(amount, rate):
    finalAmount = amount
    years = 0
    while finalAmount < 2 * amount:
        finalAmount = finalAmount * (1 + rate)
        years = years + 1
    return years
```

5.5.6
```
def hailstone(start):
    number = start
    count = 1
    print(number)
    while number > 1:
        if number % 2 == 0:
            number = number / 2
        else:
            number = 3 * number + 1
        print(number)
        count = count + 1
    return count
```

5.5.7
```
def guessingGameMC(maxNumber, strategy, trials):
    totalGuesses = 0
    for trial in range(trials):
        totalGuesses = totalGuesses + strategy(maxNumber)
    return totalGuesses / trials

def plot(trials):
    list1 = []
    list2 = []
    list3 = []
```

```
        for maxNumber in range(5, 101, 5):
            avgGuesses = guessingGameMC(maxNumber, guessingGame1, trials)
            list1.append(avgGuesses)
            avgGuesses = guessingGameMC(maxNumber, guessingGame2, trials)
            list2.append(avgGuesses)
            avgGuesses = guessingGameMC(maxNumber, guessingGame3, trials)
            list3.append(avgGuesses)

        pyplot.plot(range(5, 101, 5), list1, label = 'Strategy 1')
        pyplot.plot(range(5, 101, 5), list2, label = 'Strategy 2')
        pyplot.plot(range(5, 101, 5), list3, label = 'Strategy 3')
        pyplot.legend(loc = 'upper center')
        pyplot.xlabel('Maximum secret number')
        pyplot.ylabel('Average number of guesses')
        pyplot.show()

    plot(10000)
```

5.2 PROJECT SOLUTIONS

Project 5.1

```
import random
import math
import matplotlib.pyplot as pyplot

def poll(percentage, pollSize):
    count = 0
    for person in range(pollSize):
        if random.random() < percentage / 100:
            count = count + 1
    return count / pollSize * 100

def pollExtremes(percentage, pollSize, trials):
    sampleList = []
    for trial in range(trials):
        sampleList.append(poll(percentage, pollSize))
    return min(sampleList), max(sampleList)

def plotResults(percentage, minPollSize, maxPollSize, step, trials):
    lows = []
    highs = []
    pollSizes = range(minPollSize, maxPollSize + 1, step)
    for pollSize in pollSizes:
        low, high = pollExtremes(percentage, pollSize, trials)
        lows.append(low)
        highs.append(high)

    pyplot.axhline(percentage, color = 'red')
    pyplot.plot(pollSizes, lows, color = 'blue')
    pyplot.plot(pollSizes, highs, color = 'blue')
    pyplot.xlim(minPollSize - step, maxPollSize + step)
    pyplot.ylim(percentage - 20, percentage + 20)
    pyplot.xlabel('Poll size')
```

```
    pyplot.ylabel('Range of responses')
    pyplot.show()

    return (high - low) / 2

def plotErrors(pollSize, minPercentage, maxPercentage, step, trials):
    errors = []
    percentages = range(minPercentage, maxPercentage, step)
    for percentage in percentages:
        low, high = pollExtremes(percentage, pollSize, trials)
        errors.append((high - low) / 2)

    pyplot.plot(percentages, errors)
    pyplot.xlabel('Percentage')
    pyplot.ylabel('Margin of error')
    pyplot.show()


def main():
    error = plotResults(5, 100, 3000, 100, 1000)
    print(error)
    plotErrors(1000, 5, 95, 5, 1000)

main()
```

Project 5.2

```
import random, math, turtle
import matplotlib.pyplot as pyplot

def distance(x1, y1, x2, y2):
    return math.sqrt((x1 - x2)**2 + (y1 - y2)**2)

def angle(x, y):
    if x == 0:       # avoid divide by zero
        x = 0.001
    angle = math.degrees(math.atan(y / x))
    if angle < 0:
        if y < 0:
            angle = angle + 360   # quadrant IV
        else:
            angle = angle + 180   # quadrant II
    elif y < 0:
        angle = angle + 180       # quadrant III
    return angle

def setupWalls(tortoise, openingDegrees, scale, radius):
    screen = tortoise.getscreen()
    screen.mode("logo")                     # east is 0 degrees
    screen.tracer(5)                        # speed up drawing

    tortoise.up()                           # draw boundary with
    tortoise.width(0.015 * scale)           #   shaded background
    tortoise.goto(radius * scale, 0)
```

```
    tortoise.down()
    tortoise.pencolor("lightyellow")
    tortoise.fillcolor("lightyellow")
    tortoise.begin_fill()
    tortoise.circle(radius * scale)
    tortoise.end_fill()
    tortoise.pencolor("black")
    tortoise.circle(radius * scale, 360 - openingDegrees)
    tortoise.up()
    tortoise.home()

    tortoise.pencolor("blue")         # particle is a blue circle
    tortoise.fillcolor("blue")
    tortoise.shape("circle")
    tortoise.shapesize(0.75, 0.75)

    tortoise.width(1)             # set up for walk
    tortoise.pencolor("green")
    tortoise.speed(0)
    tortoise.down()               # comment this out to hide trail


def escape(openingDegrees, tortoise, draw):
    # opening is between 360 - openingDegrees and 360

    x = 0                         # initialize (x, y) = (0, 0)
    y = 0
    radius = 1                    # moving in unit radius circle
    stepLength = math.pi / 128    # std dev of each step

    if draw:
        scale = 300               # scale up drawing
        setupWalls(tortoise, openingDegrees, scale, radius)

    steps = 0                     # number of steps so far
    escaped = False               # has particle escaped yet?
    while not escaped:
        prevX = x
        prevY = y

        # distance moved in each direction is normal with mean = 0
        # and standard deviation = step length

        x = x + random.gauss(0, stepLength)
        y = y + random.gauss(0, stepLength)

        steps = steps + 1

        if distance(0, 0, x, y) >= radius:
            particleAngle = angle(x, y)

            if particleAngle >= 360 - openingDegrees:  # and particleAngle <= 360
                escaped = True
            else:
```

```
                x = prevX
                y = prevY

            if draw:
                if escaped:                     # make particle escape
                    x = x + 10 * (x - prevX)    # in current direction
                    y = y + 10 * (y - prevY)
                    tortoise.color("red")
                tortoise.goto(x * scale, y * scale)  # move particle

        if draw:
            screen = tortoise.getscreen()   # update screen to compensate
            screen.update()                 #  for high tracer value

        return steps

    def escapeMonteCarlo(openingDegrees, trials):
        totalSteps = 0
        for trial in range(trials):
            steps = escape(openingDegrees, None, False)
            totalSteps = totalSteps + steps
        return totalSteps / trials

    def plotEscapeSteps(minOpening, maxOpening, openingStep, trials):
        stepsList = []
        for opening in range(minOpening, maxOpening + 1, openingStep):
            steps = escapeMonteCarlo(opening, trials)
            stepsList.append(steps * (math.pi/128)**2)
            print(opening, steps)

        pyplot.plot(range(minOpening, maxOpening + 1, openingStep), stepsList,
                    label = "Simulation")
        y = [1/2 - 2*math.log(math.sin(((a/360)*math.pi)/2)) \
             for a in range(minOpening, maxOpening + 1, openingStep)]
        pyplot.plot(range(minOpening, maxOpening + 1, openingStep), y, label = "Theoretical")
        pyplot.legend(loc = "center right")
        pyplot.xlabel("Opening angle")
        pyplot.ylabel("Time to escape")
        pyplot.show()
```

CHAPTER 6

Text, documents, and DNA

6.1 EXERCISE SOLUTIONS

Section 6.1

6.1.1
```
def twice(text):
    return text + ' ' + text
```

6.1.2
```
def repeat(text, n):
    return text * n
```

6.1.3
```
def vowels(word):
    word = word.upper()
    return word.count('A') + word.count('E') + word.count('I') + \
           word.count('O') + word.count('U')
```

6.1.4
```
def nospaces(sentence):
    return sentence.replace(' ', '_')
```

6.1.5
```
def txtHelp(txt):
    newTxt = txt.replace('brb', 'be right back')
    newTxt = newTxt.replace('lol', 'laugh out loud')
    newTxt = newTxt.replace('imo', 'in my opinion')
    newTxt = newTxt.replace('u ', 'you ')
    newTxt = newTxt.replace('r ', 'are ')
    return newTxt
```

6.1.6
```
def letters(text):
    for letter in text:
        print(letter)
```

6.1.7
```
def count(text, letter):
    tally = 0
    for character in text:
        if character == letter:
            tally = tally + 1
    return tally
```

6.1.8
```
def vowels(word):
    word = word.upper()
    count = 0
    for char in word:
        if char in 'AEIOU':
            count = count + 1
    return count
```

6.1.9
```
def sentences(text):
    count = 0
    prevChar = ''
    for char in text:
        if char in '.!?' and prevChar not in '.!?':
            count = count + 1
        prevChar = char
    return count
```

6.1.10
```
def nospaces(sentence):
    new = ''
    for char in sentence:
        if char == ' ':
            new = new + '_'
        else:
            new = new + char
    return new
```

6.1.11
```
def evenParity(bits):
    return bits.count('1') % 2 == 0
```

```
def evenParity(bits):
    ones = 0
    for bit in bits:
        if bit == '1':
            ones = ones + 1
    return ones % 2 == 0

def makeEvenParity(bits):
    if evenParity(bits):
        return bits + '0'
    return bits + '1'
```

Section 6.2

6.2.1
```
def lineNumbers(fileName):
    textFile = open(fileName, 'r', encoding = 'utf-8')
    lineCount = 1
    for line in textFile:
        if (lineCount - 1) % 10 == 0:
            print('{0:<5} {1}'.format(lineCount, line[:-1]))
        else:
            print('      ' + line[:-1])
        lineCount = lineCount + 1
    textFile.close()
```

6.2.2
```
import urllib.request as web

def wordCount5(text):
    """ (docstring omitted) """

    count = 0
    prevCharacter = ' '
    for character in text:
        if character in ' \n\t' and prevCharacter not in ' \n\t':
            count = count + 1
```

```
            prevCharacter = character
        if prevCharacter not in ' \n\t':
            count = count + 1
        return count

    def wcWeb(url):
        textFile = web.urlopen(url)
        text = textFile.read()
        text = text.decode('utf-8')
        textFile.close()

        return wordCount5(text)
```

6.2.3
```
def wcLines(fileName):
    textFile = open(fileName, 'r', encoding = 'utf-8')
    lineCount = 1
    for line in textFile:
        wc = wordCount5(line)
        print('Line ' + str(lineCount) + ':', wc, 'words')
        lineCount = lineCount + 1
    textFile.close()
```

6.2.4
```
def piglatin(word):
   return word[1:] + word[0] + 'ay'

def pigLatinDict(fileName):
    textFile = open(fileName, 'r', encoding = 'utf-8')
    for line in textFile:
        pig = piglatin(line.rstrip())
        print(pig)
    textFile.close()
```

6.2.5
```
def piglatin(word):
   return word[1:] + word[0] + 'ay'

def pigLatinDict(fileName, newFileName):
    textFile = open(fileName, 'r', encoding = 'utf-8')
    newTextFile = open(newFileName, 'w')
    for line in textFile:
        pig = piglatin(line.rstrip())
        newTextFile.write(pig + '\n')
    textFile.close()
    newTextFile.close()
```

6.2.6
```
def strip(fileName, newFileName):
    textFile = open(fileName, 'r', encoding = 'utf-8')
    newTextFile = open(newFileName, 'w')
    for line in textFile:
        line = line.replace(' ', '')
        line = line.replace('\n', '')
        line = line.replace('\t', '')
        newTextFile.write(line)
    textFile.close()
    newTextFile.close()
```

Section 6.3

6.3.1 (a) `len(word)`
(b) `word[0]`
(c) `word[2]`
(d) `word[-1]`
(e) `word[-3:]`
(f) `word[1:4]`
(g) `word[-5:-2]`
(h) `word[:-1]`

6.3.2
```
def username(first, last):
    return last + '_' + first[0]
```

6.3.3
```
def piglatin(word):
   return word[1:] + word[0] + 'ay'
```

6.3.4 (a) `quote[5:9]`
(b) `quote[25:]`
(c) `quote[6:16]`
(d) `quote[:7]`

6.3.5
```
def noVowels(text):
    newText = ''
    for letter in text:
        if letter not in 'aeiouAEIOU':
            newText = newText + letter
    return newText
```

6.3.6
```
def encode(word):
    even = ''
    odd = ''
    for index in range(len(word)):
        if index % 2 == 0:
            even = even + word[index]
        else:
            odd = odd + word[index]
    return even + odd
```

6.3.7
```
def decode(codeword):
    mid = (len(codeword) + 1) // 2
    even = codeword[:mid]
    odd = codeword[mid:]
    word = ''
    for index in range(mid):
        word = word + even[index]
        if index < len(odd):
            word = word + odd[index]
    return word
```

6.3.8
```
def daffy(word):
    newWord = ''
    for letter in word:
        if letter == 's':
            newWord = newWord + 'sth'
        else:
            newWord = newWord + letter
    return newWord
```

6.3.9
```
import random

def randomPlate(length):
    plate = ''
    for index in range(length):
        plate = plate + chr(ord('A') + random.randrange(26))
    plate = plate + ' '
    for index in range(length):
        plate = plate + chr(ord('0') + random.randrange(10))
    return plate
```

6.3.10
```
def int2String(value):
    intString = ''
    while value > 0:
        digit = value % 10
        value = value // 10
        intString = digit2String(digit) + intString
    return intString
```

6.3.11
```
def reverse(text):
    revText = ''
    for character in text:
        revText = character + revText
    return revText
```

6.3.12
```
def married(fullName, spouseLastName, hyphenate):
if hyphenate:
return fullName + '-' + spouseLastName
else:
newName = ''
index = 0
while (fullName[index] != ' '):
newName = newName + fullName[index]
index = index + 1
return newName + ' ' + spouseLastName
```

6.3.13
```
def index2Letter(index):
    if (index < 1) or (index > 26):
        return None
    return chr(ord('A') + index - 1)
```

6.3.14
```
def digit2Value(digit):
    if (digit < '0') or (digit > '9'):
        return None
    return ord(digit) - ord('0')
```

6.3.15
```
def grade2Letter(grade):
    if (grade < 0) or (grade > 100):
        return None
    if grade == 100:
        return 'A'
    if grade < 60:
        return 'F'
    return chr(9 - int(grade / 10) + ord('A'))
```

6.3.16
```
def capitalize(word):
    if word[0] >= 'A' and word[0] <= 'Z':
        return word
    return chr(ord(word[0]) - ord('a') + ord('A')) + word[1:]
```

6.3.17
```
def checksum(word):
    sum = 0
    for letter in word:
        sum = sum + ord(letter) - ord('a')
    sum = sum % 26
    return word + chr(sum + ord('a'))
```

6.3.18
```
def checksumCheck(word):
    sum = 0
    for letter in word[:-1]:
        sum = sum + ord(letter) - ord('a')
    sum = sum % 26
    return word[-1] == chr(sum + ord('a'))
```

6.3.19
```
def encipher(text, shift):
    cipherText = ''
    for char in text:
        if char >= 'A' and char <= 'Z':
            cipherChar = chr(ord('A') + (ord(char) - ord('A') + shift) % 26)
        else:
            cipherChar = char
        cipherText = cipherText + cipherChar
    return cipherText
```

6.3.20
```
def encipher(text, shift, encrypt):
    if encrypt:
        shift = -shift
    cipherText = ''
    for char in text:
        if char >= 'A' and char <= 'Z':
            cipherChar = chr(ord('A') + (ord(char) - ord('A') + shift) % 26)
        else:
            cipherChar = char
        cipherText = cipherText + cipherChar
    return cipherText
```

Section 6.4

6.4.1 (a) 13

(b) 4

(c) 20

(d) 14

(e) 20

(f) $3(n+1)+2$

(g) $3(m+1)+2$

(h) When `index < len(word1)` or `index < len(word2)` is false, the following comparison(s) in the `while` loop condition are not executed due to short circuit evaluation of the `and` operator.

6.4.2 (a) linear time

(b) linear time

(c) constant time

(d) linear time

(e) linear time (because `name` must be copied to the new string)

(f) constant time

(g) constant time

(h) linear time

(i) constant time

(j) linear time

(k) constant time

(l) linear time

(m) quadratic time (because there are n calls to the `find` method, each of which is linear time)

6.4.3 (a) linear

(b) constant

(c) quadratic

(d) exponential

(e) $n \log_2 n$

6.4.4

n	$12n$		n^2	
	Steps	Time	Steps	Time
10	120	0.00000012 sec	100	0.0000001 sec
10^2	1,200	0.0000012 sec	10,000	0.00001 sec
10^4	120,000	0.00012 sec	10^8	0.1 sec
10^6	12,000,000	0.012 sec	10^{12}	$1,000$ sec $=$ 16.67 min
10^9	12 billion	12 sec	10^{18}	10^9 sec $=$ 31.71 yrs

Section 6.5

6.5.1 (a)
```
for index in range(len(text)):
    print(text[index])
```

(b)
```
newText = ''
for index in range(len(text)):
    if text[index] != ' ':
        newText = newText + text[index]
```

(c)
```
for index in range(2, min(10, len(text))):
    if text[index] >= 'a' and text[index] <= 'z':
        print(text[index])
```

(d)
```
for index in range(1, len(text) - 1)
    print(text.count(text[index]))
```

6.5.2 (a) missing `caps = ''` before loop

(b) missing initialization of `answer` before loop

(c) should be `range(len(text))`

6.5.3
```
def prefixes(word):
    for index in range(len(word)):
        print(word[:index])
```

6.5.4
```
def findWord(text, target):
    for index in range(len(text) - len(target) + 1):
        if (text[index:index + len(target)] == target) and \
           (index == 0 or text[index - 1] in ' ') and \
           (index + len(target) == len(text) - 1 or
            text[index + len(target)] in ' .!?;-,\'"'):
            return index
    return -1
```

6.5.6
```
def concordance(dictFileName, textFileName):
    dictFile = open(dictFileName, 'r', encoding = 'utf-8')
    for line in dictFile:
        concordance(textFileName, line.rstrip())
        print()
    dictFile.close()

concordance('/usr/share/dict/words', 'mobydick.txt')
```

6.5.7
```
def findReason(product):
    index = 0
    while product[index:index + 6] != '<DATE>':
        index = index + 1

    year = product[index + 18: index + 22]

    index = index + 44
    while product[index:index + 8] != '<REASON>':
        index = index + 1

    reason = ''
    index = index + 17
    while product[index:index + 12] != ']]></REASON>':
        reason = reason + product[index]
        index = index + 1

    return year + ' ' + reason

def recalls(reason, year):
    url = 'http://www.fda.gov/DataSets/Recalls/RecallsDataSet.xml'
    webpage = web.urlopen(url)

    line = ''
    while line[:14] != '<RECALLS_DATA>':
        line = webpage.readline()
        line = line.decode('utf-8')

    count = 0                               # NEW
    line = webpage.readline()
    line = line.decode('utf-8')
    while line[:15] != '</RECALLS_DATA>':
        product = ''                        # NEW
        while line[:10] != '</PRODUCT>':
            product = product + line        # NEW
            line = webpage.readline()
            line = line.decode('utf-8')
```

```
            product = product + line          # NEW
            result = findReason(product)      # NEW
            if reason in result[5:] and \
               result[:4] == str(year):       # NEW
                    count = count + 1         # NEW
            line = webpage.readline()
            line = line.decode('utf-8')
        webpage.close()

        return count                          # NEW
```

Section 6.6

6.6.1
```
i
im
imh
imho
m
mh
mho
h
ho
o
```

6.6.2
```
def difference(word1, word2):
    index = 0
    while index < len(word1) and index < len(word2) and word1[index] == word2[index]:
        index = index + 1
    if len(word1) == len(word2) and index == len(word1):
        return -1
    return index
```

6.6.3
```
def hamming(bits1, bits2):
    distance = 0
    for index in range(len(bits1)):
        if bits1[index] != bits2[index]:
            distance = distance + 1
    return distance
```

6.6.4
```
def hamming(bits1, bits2):
    distance = 0
    length = min(len(bits1), len(bits2))
    for index in range(length):
        if bits1[index] != bits2[index]:
            distance = distance + 1
    distance = distance + (max(len(bits1), len(bits2)) - length)
    return distance
```

6.6.5
```
def dotplot(text1, text2, n):  # window size
    text1 = text1.lower()
    text2 = text2.lower()
    x = []
    y = []
    for index1 in range(len(text1) - n + 1):
        for index2 in range(len(text2) - n + 1):
            if text1[index1:index1 + n] == text2[index2:index2 + n]:
                x.append(index1)
```

```
                    y.append(index2)

        pyplot.scatter(x, y)
        pyplot.xlim(0, len(text1))
        pyplot.ylim(0, len(text2))
        pyplot.show()
```

6.6.6
```
import matplotlib.pyplot as pyplot

def wordFrequency(fileName, word, window, skip):
    inputFile = open(fileName, 'r', encoding = 'utf-8')
    text = inputFile.read()
    inputFile.close()

    freqs = []
    indices = range(0, len(text), skip)
    for index in indices:
        freqs.append(text[index:index + windowSize].count(word))

    pyplot.scatter(indices, freqs, marker = '.')
    pyplot.xlabel('Location')
    pyplot.ylabel('Count')
    pyplot.xlim(0, len(text))
    pyplot.ylim(0, max(freqs))
    pyplot.show()

wordFrequency('mobydick.txt', 'ship', 2000, 1000)
```

Section 6.7

6.7.1
```
def countACG(dna):
    dna = dna.lower()
    count = 0
    for nt in dna:
        if nt != 't':
            count = count + 1
    return count / len(dna)
```

6.7.2
```
def countACG(dna):
    dna = dna.lower()
    count = 0
    for index in range(len(dna)):
        if dna[index] != 't':
            count = count + 1
    return count / len(dna)
```

6.7.3
```
def printCodons(dna):
    for index in range(0, len(dna) - 2, 3):
        print(dna[index:index+3])
```

6.7.4
```
def findCodon(dna, codon):
    """Find the index of the first occurrence of a codon in a
       DNA sequence.

    Parameters:
        dna: a string representing a DNA sequence
        codon: a string representing a codon
```

```
        Return value: the index of the first occurrence of the codon in dna
        """

        for index in range(len(dna) - 2):
            if dna[index:index + 3] == codon:
                return index
        return -1
```

6.7.5
```
def findATG(dna):
    dna = dna.lower()
    positions = []
    for index in range(len(dna)):
        if dna[index:index+3] == 'atg':
            positions.append(index)
    return positions
```

6.7.6
```
def printCodonsAll(dna):
    for rf in range(3):
        print()
        print('Reading frame', rf +1, ':')
        printCodons(dna[rf:])
```

6.7.7
```
def CpG(dna):
    dna = dna.lower()
    count = 0
    for index in range(0, len(dna)):
        if dna[index:index+2] == 'cg':
            count = count + 1
    return count * 2 / len(dna)
```

6.7.8
```
def ssr(dna, repeat):
    start = dna.find(repeat)
    if start < 0:
        return 0
    count = 0
    for index in range(start, len(dna), len(repeat)):
        if dna[index : index + len(repeat)] == repeat:
            count = count + 1
        else:
            return count
```

OR

```
def ssr(dna, repeat):
    start = dna.find(repeat)
    if start < 0:
        return 0
    count = 0
    index = start
    while index < len(dna) and dna[index:index + len(repeat)] == repeat:
        count = count + 1
        index = index + len(repeat)
    return count
```

6.7.9
```
def ssr2(dna, repeat):
    longest = 0
    start = 0
```

```
        found = 1
        while found > 0 and start < len(dna) - longest:
            sub = dna[start:]
            found = ssr(sub, repeat)
            if found > 0:
                longest = max(longest, found)
                start = start + found * len(repeat)
        return longest
```

6.7.10
```
def ssr3(dna):
    longest = 0
    for b in 'cgat':
        for c in 'cgat':
            repeat = b + c
            count = ssr2(dna, repeat)
            if count > longest:
                longest = count
                longestRepeat = repeat
    return longestRepeat
```

6.7.11
```
def palindrome(dna):
    return dna == reverseComplement(dna)
```

6.7.12
```
def dna2rna(dna):
    dna = dna.lower()
    rna = ''
    for nt in dna:
        if nt == 't':
            rna = rna + 'u'
        else:
            rna = rna + nt
    return rna
```

6.7.13
```
def transcribe(dna):
    return dna2rna(reverse(complement(dna)))
```

6.7.14
```
def clean(dna):
    dna = dna.lower()
    cleanDNA = ''
    for nt in dna:
        if nt not in 'acgt':
            cleanDNA = cleanDNA + 'n'
        else:
            cleanDNA = cleanDNA + nt
    return cleanDNA
```

6.7.15
```
def fix(dna):
    dna = dna.lower()
    fixDNA = ''
    for nt in dna:
        if nt == 'r':
            if random.random() < 0.5:
                base = 'a'
            else:
                base = 'g'
        elif nt == 'y':
            if random.random() < 0.5:
                base = 'c'
```

```
    else:
        base = 't'
elif nt == 'k':
    if random.random() < 0.5:
        base = 'g'
    else:
        base = 't'
elif nt == 'm':
    if random.random() < 0.5:
        base = 'a'
    else:
        base = 'c'
elif nt == 's':
    if random.random() < 0.5:
        base = 'c'
    else:
        base = 'g'
elif nt == 'w':
    if random.random() < 0.5:
        base = 'a'
    else:
        base = 't'
elif nt == 'b':
    choice = random.randrange(3)
    if choice == 0:
        base = 'c'
    elif choice == 1:
        base = 'g'
    else:
        base = 't'
elif nt == 'd':
    choice = random.randrange(3)
    if choice == 0:
        base = 'a'
    elif choice == 1:
        base = 'g'
    else:
        base = 't'
elif nt == 'h':
    choice = random.randrange(3)
    if choice == 0:
        base = 'a'
    elif choice == 1:
        base = 'c'
    else:
        base = 't'
elif nt == 'v':
    choice = random.randrange(3)
    if choice == 0:
        base = 'a'
    elif choice == 1:
        base = 'c'
    else:
        base = 'g'
```

```
            elif nt == 'n':
                choice = random.randrange(4)
                if choice == 0:
                    base = 'a'
                elif choice == 1:
                    base = 'c'
                elif choise == 2:
                    base = 'g'
                else:
                    base = 't'
            else:
                base = nt
            fixDNA = fixDNA + base
```

6.7.16
```
def mark(dna):
    dna = dna.lower()
    newDNA = ''
    for index in range(0, len(dna), 3):
        if dna[index:index+3] == 'atg':
            newDNA = newDNA + '>>>'
        elif dna[index:index+3] == 'taa' or dna[index:index+3] == 'tag' or \
             dna[index:index+3] == 'tga':
            newDNA = newDNA + '<<<'
        else:
           newDNA = newDNA + dna[index:index+3]
    return newDNA
```

6.7.17
```
hepA = getFASTA('NC_001489')
print(gcContent(hepA))
```

6.7.18
```
hepC1 = getFASTA('NP_671491')
hepC2 = getFASTA('YP_001469630')
dotplot(hepC1, hepC2, 4)
```

6.7.19
```
human = getFASTA('U10421.1')
mouse = getFASTA('NM_010449.4')
dotplot(human, mouse, 8)
```

6.2 PROJECT SOLUTIONS

Project 6.1

```
import urllib.request as web
import matplotlib.pyplot as pyplot

def getVote(line):
    index = 0
    while line[index:index + 6] != 'party=':
        index = index + 1
    party = line[index + 7]

    while line[index:index + 6] != 'state=':
        index = index + 1
    state = line[index + 7: index + 9]

    index = index + 8
    while line[index:index + 6] != '<vote>':
```

```
        index = index + 1

    vote = ''
    index = index + 6
    while line[index:index + 7] != '</vote>':
        vote = vote + line[index]
        index = index + 1

    return state + ' ' + party + ' ' + vote

def partyLine(year, number):
    url = 'http://clerk.house.gov/evs/' + str(year) + \
          '/roll' + '{0:0>3}'.format(number) + '.xml'
    webpage = web.urlopen(url)

    rAye = 0
    rNo = 0
    rNV = 0
    dAye = 0
    dNo = 0
    dNV = 0
    for line in webpage:
        line = line.decode('utf-8')
        if line[:15] == '<recorded-vote>':
            voteRecord = getVote(line)
            party = voteRecord[3]
            vote = voteRecord[5:]
            if party == 'D':
                if vote == 'No' or vote == 'Nay':
                    dNo = dNo + 1
                elif vote == 'Aye' or vote == 'Yea':
                    dAye = dAye + 1
                else:
                    dNV = dNV + 1
            elif party == 'R':
                if vote == 'No' or vote == 'Nay':
                    rNo = rNo + 1
                elif vote == 'Aye' or vote == 'Yea':
                    rAye = rAye + 1
                else:
                    rNV = rNV + 1

    webpage.close()
    totalD = dAye + dNo
    totalR = rAye + rNo

    return ((dAye > totalD / 2) and (rNo > totalR / 2)) or \
           ((rAye > totalR / 2) and (dNo > totalD / 2))

def countPartyLine(year, maxNumber):
    partyLineCount = 0
    for number in range(1, maxNumber + 1):
        result = partyLine(year, number)
        if result:
```

```
            partyLineCount = partyLineCount + 1
    return partyLineCount / maxNumber

def plotPartyLine():
    fractions = []
    for year in range(1994, 2014):
        fraction = countPartyLine(year, 450)
        fractions.append(fraction)
        print(fraction)

    pyplot.plot(range(1994, 2014), fractions)
    pyplot.xlabel('Year')
    pyplot.ylabel('Party line votes')
    pyplot.show()

def stateDivide(state):
    years = range(1994, 2015)
    demList = []
    repList = []
    for year in years:
        url = 'http://clerk.house.gov/evs/' + str(year) + '/roll001.xml'
        webpage = web.urlopen(url)
        democrats = 0
        republicans = 0
        for line in webpage:
            line = line.decode('utf-8')
            if line[:15] == '<recorded-vote>':
                voteRecord = getVote(line)
                recordState = voteRecord[:2]
                party = voteRecord[3]
                if recordState == state:
                    if party == 'D':
                        democrats = democrats + 1
                    elif party == 'R':
                        republicans = republicans + 1
        demList.append(democrats)
        repList.append(republicans)

    pyplot.plot(years, demList, label = 'Democrats')
    pyplot.plot(years, repList, label = 'Republicans')
    pyplot.legend(loc = 'lower center')
    pyplot.xlabel('Year')
    pyplot.ylabel('Number')
    pyplot.xlim(years[0], years[-1])
    pyplot.show()
```

Project 6.2

```
import turtle

width = 1440            # width of the window
cols = width // 6       # number of columns of text
height = 600            # height of the window
rows = height // 100    # number of rows of text
```

```
def plot(tortoise, index, value, window):
    """Plot GC fraction value for window ending at position index."""

    if (index == window) or (index - window + 1) // cols != (index - window) // cols:
        tortoise.up()
        tortoise.goto((index - window + 1) % cols, \
                      (index - window + 1) // cols + 0.7 + value * 0.25)
        tortoise.down()
    else:
        tortoise.goto((index - window + 1) % cols, \
                      (index - window + 1) // cols + 0.7 + value * 0.25)

def bar(tortoise, index, rf):
    """Draw a colored bar over codon starting at position index in
       reading frame rf.  Put the turtle's tail up and down to
       handle line breaks properly."""

    tortoise.up()
    tortoise.goto(index % cols, index // cols + (rf + 1) / 5)
    tortoise.down()
    tortoise.forward(1)
    tortoise.up()
    tortoise.goto((index + 1) % cols, (index + 1) // cols + (rf + 1) / 5)
    tortoise.down()
    tortoise.forward(1)
    tortoise.up()
    tortoise.goto((index + 2) % cols, (index + 2) // cols + (rf + 1) / 5)
    tortoise.down()
    tortoise.forward(1)

def gcFreq(dna, window, tortoise):
    """Plot GC frequency over a sliding window."""

    # draw red lines at 0.5 above the sequence

    tortoise.pencolor('red')
    for index in range(len(dna) // cols + 1):
        tortoise.up()
        tortoise.goto(0, index + 0.825)
        tortoise.down()
        if index < len(dna) // cols:
            tortoise.goto(cols - 1, index + 0.825)
        else:
            tortoise.goto((len(dna) - window) % cols, index + 0.825)

    tortoise.pencolor('blue')

    # get initial window count
    count = 0
    for base in dna[:window]:
        if base == 'C' or base == 'G':
            count = count + 1
```

```
    # get subsequent window counts and plot them
    for index in range(window, len(dna)):
        outbase = dna[index - window]
        inbase = dna[index]
        if outbase == 'C' or outbase == 'G':
            count = count - 1
        if inbase == 'C' or inbase == 'G':
            count = count + 1

        # plot value for window ending at position index
        plot(tortoise, index, count / window, window)

def orf1(dna, rf, tortoise):
    """Find all ORFs in reading frame rf = 0, 1, 2 (forward only),
       not including ORFs contained in other ORFs."""

    startCodon = 'ATG'
    stopCodons = ['TAA', 'TAG', 'TGA']

    tortoise.pencolor('red')
    inORF = False
    for index in range(rf, len(dna) - 2, 3):
        if not inORF and dna[index:index+3] == startCodon: # found start of new ORF
            inORF = True
            tortoise.pencolor('blue')
            bar(tortoise, index, rf)
        elif inORF and dna[index:index+3] in stopCodons:  # found end of an ORF
            inORF = False
            bar(tortoise, index, rf)
            tortoise.pencolor('red')
        else:
            bar(tortoise, index, rf)

def viewer(dna):
    """Display GC content and ORFs in 3 forward reading frames."""

    dna = dna.upper()      # make everything upper case

    tortoise = turtle.Turtle()
    screen = tortoise.getscreen()
    screen.setup(width, height)                # make a long, thin window
    screen.setworldcoordinates(0, 0, cols, rows) # scale so 1 char fits at each point
    screen.tracer(100)
    tortoise.hideturtle()
    tortoise.speed(0)
    tortoise.up()

    # Draw DNA string in window.

    for index in range(len(dna)):
        tortoise.goto(index % cols, index // cols)
        tortoise.write(dna[index], font = ('Courier', 9, 'normal'))

    # Find ORFs in forward reading frames 0, 1, 2.
```

```
    tortoise.width(5)
    for index in range(3):
        orf1(dna, index, tortoise)

    # Plot GC frequency.

    tortoise.width(1)
    gcFreq(dna, 5, tortoise)

    screen.update()
    screen.exitonclick()

def main():
    # Read DNA from a file and find ORFs

    inputFile = open('Eco536-1K.txt', 'r')
    dna = inputFile.read()
    viewer(dna)

main()
```

Designing programs

7.1 EXERCISE SOLUTIONS

Section 7.2

7.2.4
```
def volumeSphere(radius):
    """Computes the volume of a sphere.

    Precondition: radius is positive

    Postcondition: returns the volume of a sphere with
                   the given radius
    """

    assert radius > 0, 'radius must be positive'
    return (4 / 3) * math.pi * (radius ** 3)
```

7.2.5
```
def fair(employee, ceo, ratio):
    """Determines whether CEO salary is fair relative
       to the average employee.

    Precondition: employee is positive

    Postcondition: returns a Boolean indicating whether the
                   CEO salary is fair
    """

    assert employee > 0, 'employee must have a salary!'
    return (ceo / employee <= ratio)
```

7.2.6
```
def windChill(temp, wind):
    """Gives the North American metric wind chill equivalent
       for given temperature and wind speed.

    Preconditions:
        temp is a numeric temperature (<= 10) in degrees Celsius
        wind is a numeric wind speed (>= 4.8) at 10m in km/h

    Postcondition:
        returns the equivalent wind chill in degrees Celsius, rounded to
        the nearest integer
    """
```

```
        assert isinstance(temp, int) or isinstance(temp, float), \
               'temperature must be a number'
        assert temp <= 10, 'temperature must be at most 10 degrees'
        assert isinstance(wind, int) or isinstance(wind, float), \
               'wind speed must be a number'
        assert wind >= 4.8, 'wind speed must be at least 4.8 km/h'

        chill = 13.12 + 0.6215 * temp + (0.3965 * temp - 11.37) * wind ** 0.16
        return round(chill)
```

7.2.7

```
def plot(tortoise, n):
    """Plots x**2 from x = -n to n.

    Preconditions:
        tortoise is a Turtle object
        n is a positive integer representing the
          positive and negative extent of x values

    Postcondition: plots x**2 in turtle graphics between
                   x = -n and x = n, inclusive
    """

    assert isinstance(tortoise, turtle.Turtle), 'tortoise must be a Turtle object'
    assert isinstance(n, int), 'n must be an integer'
    assert n > 0, 'n must be a positive integer'

    for x in range(-n, n + 1):
        tortoise.goto(x, x * x)
```

7.2.8

```
def pond(years):
    """Simulates a fish population in a fishing pond, and
       prints annual population size.  The population
       grows 8% per year with an annual harvest of 1500.

    Precondition: years is a positive integer

    Postcondition: printed a table of annual population sizes
                   for the given number of years and
                   returned final population size
    """

    assert isinstance(years, int), 'years must be an integer'
    assert years > 0, 'years must be a positive integer'

    population = 12000
    print('Year  Population')
    print('{0:<4}  {1:>9.2f}'.format(0, population))
    for year in range(1, years + 1):
        population = 1.08 * population - 1500
        print('{0:<4}  {1:>9.2f}'.format(year, population))
    return population
```

7.2.9

```
def decayC14(originalAmount, years, dt):
    """Returns the final amount of carbon-14 after decaying
       for the given number of years.
```

```
            Precondition: originalAmount is positive number,
                          years is a positive number, and
                          dt is a float between 0 and 1

            Postcondition: returns the final amount of carbon-14
            after decaying for the given number of years
        """

        assert originalAmount > 0
        assert years >= 0
        assert dt > 0 and dt < 1
        assert isinstance(originalAmount, float) or isinstance(originalAmount, int)
        assert isinstance(years, float) or isinstance(years, int)
        assert isinstance(dt, float)

        amount = originalAmount
        k = -0.00012096809434
        numIterations = int(years / dt) + 1
        for i in range(1, numIterations):
            amount = amount + k * amount * dt
        return amount
```

7.2.10

```
def argh(n):
    """Says ar...gh like a pirate!

       Precondition: n is a positive integer

       Postcondition: prints ar...gh
    """

    assert isinstance(n, int), 'n must be an integer'
    assert n > 0, 'n must be a positive integer'
    return 'A' + ('r' * n) + 'gh!'
```

7.2.11

```
def reverse(word):
    """Returns the word reversed.

    Precondition: word is a string

    Postcondition: returns the word reversed
    """

    assert isinstance(word, str), 'word must be a string'

    newstring = ''
    for index in range(len(word) - 1, -1, -1):
        newstring = newstring + word[index]
    return newstring
```

7.2.12

```
def find(text, target):
    """Find the index of the first occurrence of target in text.

    Preconditions:
        text is a non-empty a string object to search in
        target is a non-empty string object to search for
```

```
            text must be at least as long as target

        Postcondition: returns the index of the first occurrence of target in text
        """

        assert isinstance(text, str) and len(text) > 0, \
               'first argument must be a non-empty string'
        assert isinstance(target, str) and len(target) > 0, \
               'second argument must be a non-empty string'
        assert len(text) >= len(target), \
               'text must be at least as long as target'

        for index in range(len(text) - len(target) + 1):
            if text[index:index + len(target)] == target:
                return index
        return -1
```

7.2.13
```
def lineNumbers(fileName):
    """Print the contents of the file with the given name
       with each line preceded by a line number.

    Precondition: fileName is a string with name of a text file

    Postcondition: the contents of the file are printed and
                   each line is preceded by a line number
    """

    assert isinstance(fileName, str), 'file name must be a string'
    assert os.path.isfile(fileName), 'file does not exist'
    assert os.access(fileName, os.R_OK), 'file cannot be read'

    textFile = open(fileName, 'r', encoding = 'utf-8')
    lineCount = 1
    for line in textFile:
        print('{0:<5} {1}'.format(lineCount, line[:-1]))
        lineCount = lineCount + 1
    textFile.close()
```

Section 7.3

7.3.2
```
def test_digit2String():
    assert digit2String(7) == '7' #common cases
    assert digit2String(0) == '0'
    assert digit2String(1) == '1'
    for i in range(1,9):
        assert digit2String(i) == str(i)

    print('Passed all tests of digit2String!')
```

7.3.3
```
def test_int2String():
    assert int2String(9) == '9' #common cases
    for i in range(1, 100 ):
        assert int2String(i ) == str(i)
    assert int2String(10000000000) == '10000000000'
```

```
        print('Passed all tests of int2String')
```

7.3.4
```
def test_assignGP():
    assert assignGP(90) == 4
    assert assignGP(10000) == 4  # should fix this case to be invalid input
    assert assignGP(-1) == 0
    assert assignGP(80) == 3
    assert assignGP(75) == 2
```

7.3.5
```
def test_volumeSphere():
    assert volumeSphere(10) > 4188.789 and volumeSphere(10) < 4188.791
    assert volumeSphere(1) > 4.18879 and volumeSphere(1) < 4.18881
    assert volumeSphere(2) > 33.51032 and volumeSphere(2) < 33.51035
    assert volumeSphere(100) > 4188790.20478 and volumeSphere(100) < 4188790.20481
    assert volumeSphere(0) == 0
    assert volumeSphere(360) > 195432195.794 and volumeSphere(360) < 195432195.798

    print('Passed all tests of volumeSphere!')
```

7.3.6
```
def test_windChill():
    assert windChill(10, 5.0) == 10.0 #common cases
    assert windChill(1, 10.0) == -2.0
    assert windChill(3, 14.7) == -1.0
    assert windChill(5, 15.9) == 2.0
    assert windChill(-10.01, 7.3) == -14.0 #boundary cases
    assert windChill(9.9, 4.9) == 10.0

    print('Passed all tests of windChill!')
```

7.3.7
```
def test_decayC14():
    assert decayC14(100, 5000, 0.01) == 54.616134811011754
    assert decayC14(100, 20000, 0.01) == 8.897824742045865
    assert decayC14(500, 10000, 1) == 149.13530523710762
    assert decayC14(500, 10000, 0.02) == 149.14599995967973
    assert decayC14(100, 5000, 0.001) == 54.61615279333115
    assert decayC14(100, 5000, 0.1) == 54.61595498674756
    assert decayC14(1000, 500, 0.01) == 941.3087510015464
    assert decayC14(0,0, 0.001) == 0
    assert decayC14(0, 100000, 0.001) == 0.0
    assert decayC14(1000, 1, 0.001) == 999.879039214689
    assert decayC14(1000, 0, 0.001) == 1000
    assert decayC14(100000, 100, 0.001) == 98797.60621057947
    assert decayC14(0, 100000, 1) == 0.0
    assert decayC14(1000, 1, 1) == 999.87903190566
    assert decayC14( 1000, 1, 1) == 999.87903190566
    assert decayC14(100000, 100, 1) == 98797.53399040975

    print('Passed all test of decayC14!')
```

7.3.8
```
def test_reverse():
    assert reverse('test') == 'tset'
    assert reverse('Chicago') == 'ogacihC'
    assert reverse('') == ''
    assert reverse('e') == 'e'
    assert reverse('abcdefghijklmnopqrstuvwxyz') == 'zyxwvutsrqponmlkjihgfedcba'

    print('Passed all test of reverse!')
```

7.3.9
```
def test_find():
    quote = 'These tests will show our algorithm works.'
    assert find(quote, 'the') == -1
    assert find(quote, 'a') == 26
    assert find(quote, 'show ') == 17
    assert find(quote, 'works!') == -1
    assert find(quote, ' ') == 5
    assert find('a', '') == -1
    assert find('','hi') == -1
    assert find('e','e') == 0
    assert find('i', 'e') == -1
    assert find(quote,'.') == 41
    assert find(quote, 'Our') == -1
    assert find('test','te') == 0
    assert find( 'test','st') == 2
    assert find(quote, quote) == 0
    assert find('the', quote) == -1

    print('Passed all test of find!')
```

CHAPTER 8

Data analysis

8.1 EXERCISE SOLUTIONS

Section 8.1

8.1.1 (a) `print(len(data))`

(b) `print(data[2])`

(c) `print(data[-1])`

(d) `print(data[-3:])`

(e) `print(data[:4])`

(f) `print(data[1:4])`

8.1.2
```
def main():
    someData = getInputFromSomewhere()
    average = mean(someData)
    if average == None:
        print('The list was empty.')
    else:
        print('The mean value is', average)
```

8.1.3
```
def sum(data):
    sum = 0
    for item in data:
        sum = sum + item
    return sum
```

8.1.4
```
def sumOdds(data):
    sum = 0
    for item in data:
        if item % 2 == 1:
            sum = sum + item
    return sum
```

8.1.5
```
def countOdds(data):
    count = 0
    for item in data:
        if item % 2 == 1:
            count = count + 1
    return count
```

8.1.6
```
def multiples5(data):
    count = 0
```

```
        for item in data:
            if item % 5 == 0:
                count = count + 1
        return count
```

8.1.7
```
def countNames(words):
    count = 0
    for word in words:
        if word[0] >= 'A' and word[0] <= 'Z':
            count = count + 1
    return count
```

8.1.8
```
def percentile(data, value):
    count = 0
    for item in data:
        if item <= value:
            count = count + 1
    return 100 * count / len(data)
```

8.1.9
```
def meanSquares(data):
    sumSquares = 0
    for item in data:
        sumSquares = sumSquares + item * item
    meanSquares = sumSquares / len(data)
    return meanSquares
```

8.1.10
```
def variance(data):
    return meanSquares(data) - mean(data) ** 2
```

8.1.11
```
def max(data):
    maximum = data[0]
    for item in data[1:]:
        if item > maximum:
            maximum = item
    return maximum
```

8.1.12
```
def shortest(words):
    short = words[0]
    for word in words[1:]:
        if len(word) < len(short):
            short = word
    return short
```

8.1.13
```
def span(data):
    minimum = data[0]
    maximum = data[0]
    for item in data[1:]:
        if item > maximum:
            maximum = item
        if item < minimum:
            minimum = item
    return maximum - minimum
```

8.1.14
```
def maxIndex(data):
    max = 0
    for index in range(1, len(data)):
        if data[index] > data[max]:
            max = index
    return max
```

8.1.15
```
def secondLargest(data):
    maxI = maxIndex(data)
    if maxI != 0:
        max2 = data[0]
    else:
        max2 = data[1]
    for index in range(1, len(data)):
        if (index != maxI) and (data[index] > max2):
            max2 = data[index]
    return max2
```

8.1.16
```
def search(data, target):
    for item in data:
        if item == target:
            return True
    return False
```

8.1.17
```
def search(data, target):
    for index in range(len(data)):
        if data[index] == target:
            return index
    return -1
```

8.1.18
```
def intersect(data1, data2):
    for item in data1:
        if search(data2, item):
            return True
    return False
```

8.1.19
```
def differ(data1, data2):
    for index in range(len(data1)):
        if data1[index] != data2[index]:
            return index
    return -1
```

8.1.20 (a)
```
def checksum(data):
    sum = 0
    for value in data:
        sum = sum + value
    return data + [sum % 10]
```

(b)
```
def check(data):
    return data[-1] == checksum(data)
```

(c) Any error in which the order of two digits are swapped would not be detected.

8.1.21
```
def luhn(number):
    for i in range(len(number) - 2, -1, -2):
        number[i] = number[i] * 2

    total = 0
    for n in number:
        total = total + (n % 10) + (n // 10)

    return (total % 10 == 0)
```

Section 8.2

8.2.1
```
fruit.append('grapes')
fruit = fruit + ['grapes']
```

8.2.2
```
def squares(n):
    squaresList = []
    for index in range(1, n + 1):
        squaresList.append(index * index)
    return squaresList
```

8.2.3
```
def GetCodons(dna):
    codons = []
    for i in range(0, len(dna) - 2, 3):
        codons.append(dna[i:i+3])
    return codons
```

8.2.4
```
def square(data):
    for index in range(len(data)):
        data[index] = data[index] ** 2
```

8.2.5
```
def swap(data, i, j):
    temp = data[i]
    data[i] = data[j]
    data[j] = temp
```

8.2.6
```
def reverse(data):
    for index in range(len(data) // 2):
        swap(data, index, -(index + 1))
```

8.2.7
```
def winner(votes):
    countYea = 0
    countNay = 0
    for vote in votes:
        if vote == 'yea':
            countYea = countYea + 1
        else:
            countNay = countNay + 1
    if countYea > countNay:
        return 'yea'
    elif countNay > countYea:
        return 'nay'
    else:
        return 'tie'
```

8.2.8
```
def delete(data, index):
    if (index < 0) or (index >= len(data)):
        return data.copy()
    return data[:index] + data[index + 1:]
```

8.2.9
```
def remove(data, value):
    newData = []
    for item in data:
        if item != value:
            newData.append(item)
    return newData
```

8.2.10
```
def centeredMean(data):
    data.remove(min(data))
    data.remove(max(data))
    sum = 0
    for item in data:
        sum = sum + item
    return sum / len(data)
```

8.2.11
```
def adjust(rates):
    ratesCopy = []
    for index in range(len(rates)):
        ratesCopy.append(rates[index] - 0.01)
    return ratesCopy
```

8.2.12
```
import random
def shuffle(data):
    for i in range(100):                    # switch 100 random pairs
        x = random.randrange(len(data) - 1)     # get a random index
        y = random.randrange(x + 1, len(data))  # get another random index
        swap(data, x, y)                        # swap the items at indices x and y
```

8.2.13 In this loop, the variable name `value` is a local variable that has been assigned a numeric value from the list `unemployment`. Assigning `value` a new value does not change the original list.

8.2.14
```
def smooth(data, windowLength):
    smoothed = []

    sum = 0
    for value in data[:windowLength]:
        sum = sum + value
    smoothed.append(sum / windowLength)

    for index in range(1, len(data) - windowLength + 1):
        sum = sum - data[index - 1]
        sum = sum + data[index + windowLength - 1]
        smoothed.append(sum / windowLength)

    return smoothed
```

8.2.15
```
def median(data):
    data.sort()
    middle = len(data) // 2
    if len(data) % 2 == 1:
        return data[middle]
    else:
        return (data[middle - 1] + data[middle]) / 2
```

8.2.16
```
groceries.insert(4, 'jelly beans')
groceries.insert(1, 'donuts')
groceries.insert(0, 'bananas')
groceries.append('watermelon')

groceries.pop(7)
groceries.pop(7)
groceries.pop(0)
groceries.pop(3)
```

8.2.17
```
def sameBirthday(people):
    birthdays = []
    for person in range(people):
        birthdays.append(random.randrange(365))
    birthdays.sort()
    for index in range(1, people):
        if birthdays[index - 1] == birthdays[index]:
            return 1
```

```
        return 0

    def birthdayProblem(people, trials):
        totalShare = 0
        for trial in range(trials):
            totalShare = totalShare + sameBirthday(people)
        return totalShare / trials
```

8.2.18
```
def birthdayProblem2(trials):
    people = 1
    while birthdayProblem(people, trials) < 0.5:
        people = people + 1
    return people
```

8.2.19
```
def squares(n):
    return [index * index for index in range(1, n + 1)]
```

8.2.20
```
def GetCodons(dna):
    return [dna[i:i+3] for i in range(0, len(dna) - 2, 3)]
```

Section 8.3

8.3.1
```
def printFrequencies(frequency):
    keys = list(frequency.keys())
    keys.sort()
    print('Key     Frequency')
    for key in keys:
        print('{0:<8}{1:>4}'.format(key, frequency[key]))
```

8.3.2
```
def wordFrequency(text):
    frequency = {}
    words = text.split()
    for word in words:
        word = word.strip('.!?,:;')
        if word in frequency:
            frequency[word] = frequency[word] + 1
        else:
            frequency[word] = 1
    return frequency
```

8.3.3
```
def pmf(frequency):
    total = 0
    for key in frequency:
        total = total + frequency[key]
    pmfDict = { }
    for key in frequency:
        pmfDict[key] = frequency[key] / total
    return pmfDict
```

8.3.4
```
def wordFrequencies(fileName):
    freq = {}
    inputFile = open(fileName, 'r', encoding = 'utf-8')
    for line in inputFile:
        words = line.split()
        for word in words:
            word = word.strip('.!?,:;()\'"-')
            word = word.lower()
            if word in freq:
```

```
                freq[word] = freq[word] + 1
            else:
                freq[word] = 1
    inputFile.close()
    words = list(freq.keys())
    words.sort()
    for word in words:
        print(word + ': ' + str(freq[word]))
```

8.3.5
```
def firstLetterCount(words):
    dictionary = {}
    for word in words:
        first = word[0].lower()
        if first in dictionary:
            dictionary[first] = dictionary[first] + 1
        else:
            dictionary[first] = 1
    return dictionary
```

8.3.6
```
def firstLetterWords(words):
    dictionary = {}
    for word in words:
        first = word[0].lower()
        if first in dictionary:
            dictionary[first].append(word)
        else:
            dictionary[first] = [word]
    return dictionary
```

8.3.7
```
import matplotlib.pyplot as pyplot

def histogram(data):
    frequency = { }

    for item in data:
        if item in frequency:
            frequency[item] = frequency[item] + 1
        else:
            frequency[item] = 1

    indices = range(len(frequency))
    pyplot.bar(indices, frequency.values(), align = 'center')
    pyplot.xticks(indices, list(frequency.keys()))
    pyplot.xlabel('Data')
    pyplot.ylabel('Frequency')
    pyplot.show()
```

8.3.8
```
def bonus(salaries):
    for name in salaries:
        salaries[name] = salaries[name] * 1.05
```

8.3.9
```
def updateAges(names, ages):
    for name in names:
        if name in ages:
            ages[name] = ages[name] + 1
```

8.3.10
```
def seniorList(students, year):
    seniors = []
```

```
        for name in students:
            if students[name] == year:
                seniors.append(name)
        return seniors
```

8.3.11
```
def createDictionary():
    translate = {}
    translate['cat'] = 'atcay'
    translate['dog'] = 'ogcay'
    # etc...
    return translate

def translate():
    dictionary = createDictionary()
    word = input('Please type a word to translate (or quit): ')
    while word != 'quit':
        if word in dictionary:
            print('Translation:', dictionary[word])
        else:
            print('That word was not found.')
        word = input('Please type a word to translate (or quit): ')
```

8.3.12
```
def txtTranslate(word):
    translations = {'lol': 'laugh out loud', 'brb': 'be right back'}  # ...etc.
    if word in translations:
        return translations[word]
    else:
        return 'Word was not found.'
```

8.3.13
```
def login(passwords):
    username = input('Username: ')
    password = input('Password: ')
    while username not in passwords or passwords[username] != password:
        print('\nTry again.')
        username = input('Username: ')
        password = input('Password: ')
    print('Success!')
```

8.3.14
```
def union(dict1, dict2):
    unionDict = dict1.copy()
    for key in dict2:
        if key not in unionDict:
            unionDict[key] = dict2[key]
    return unionDict
```

8.3.15
```
def hardness(rocks):
    classes = {'soft': [], 'medium': [], 'hard': [], 'very hard': []}
    for rock in rocks:
        if rock[1] <= 3:
            classes['soft'].append(rock[0])
        elif rock[1] <= 5:
            classes['medium'].append(rock[0])
        elif rock[1] <= 8:
            classes['hard'].append(rock[0])
        else:
            classes['very hard'].append(rock[0])
    return classes
```

8.3.16
```
def complement(dna):
    comps = {'a': 't', 'c': 'g', 'g': 'c', 't': 'a'}
    dna = dna.lower()
    compdna = ''
    for nt in dna:
        compdna = compdna + comps[nt]
    return compdna
```

8.3.17
```
def profile1(sequences, index):
    freqs = {'A': 0, 'C': 0, 'G': 0, 'T': 0}
    for seq in sequences:
        base = seq[index]
        freqs[base] = freqs[base] + 1
    return freqs

def profile(sequences):
    profileList = [ ]
    length = len(sequences[0])
    for index in range(length):
        profileList.append(profile1(sequences, index))
    return profileList

def maxBase(freqs):
    maxBase = 'A'
    for base in freqs:
        if freqs[base] > freqs[maxBase]:
            maxBase = base
    return maxBase

def consensus(sequences):
    profileList = profile(sequences)
    consensus = ''
    for freqs in profileList:
        consensus = consensus + maxBase(freqs)
    return consensus
```

Section 8.4

8.4.1
```
def plotQuakes():
    url = 'http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_month.csv'
    quakeFile = web.urlopen(url)
    header = quakeFile.readline()

    longitudes = []
    latitudes = []
    depths = []
    magnitudes = []
    for line in quakeFile:
        line = line.decode('utf-8')
        row = line.split(',')
        latitudes.append(float(row[1]))
        longitudes.append(float(row[2]))
        depths.append(float(row[3]))
        magnitudes.append(float(row[4]))
```

```
quakeFile.close()

colors = []
for depth in depths:
    if depth < 10:
        colors.append('yellow')
    elif depth < 50:
        colors.append('red')
    else:
        colors.append('blue')

sizes = [mag ** 2 for mag in magnitudes]

pyplot.scatter(longitudes, latitudes, sizes, color = colors)
pyplot.show()
```

8.4.2
```
def firstLetterCount(filename):
    text = open(filename, 'r', encoding = 'utf-8')
    words = text.read()
    text.close()
    words = words.split()
    dictionary = {}
    for word in words:
        word = word.strip('.!?,:;')
        first = word[0].lower()
        if first in dictionary:
            dictionary[first] = dictionary[first] + 1
        else:
            dictionary[first] = 1
    return dictionary
```

8.4.3
```
def login(filename):
    passwordFile = open(filename, 'r', encoding = 'utf-8')
    passwords = {}
    for line in passwordFile:
        values = line.split()
        passwords[values[0]] = values[1].strip()
    passwordFile.close()

    username = input('Username: ')
    password = input('Password: ')
    while username not in passwords or passwords[username] != password:
        print('\nTry again.')
        username = input('Username: ')
        password = input('Password: ')
    print('Success!')
```

8.4.4
```
def plotPopulation():
    popFile = open('worldpopulation.txt', 'r')

    header = popFile.readline()

    years = []
    pops = []
    rates = []
    for line in popFile:
```

```
            values = line.split('\t')
            years.append(int(values[0]))
            pops.append(int(values[1]))
            if len(pops) > 1:
                rates.append((pops[-1] - pops[-2]) / pops[-2] * 100)

        pyplot.plot(years, pops)
        pyplot.ylabel('World population')
        pyplot.xlabel('Year')
        pyplot.xlim(years[0], years[-1])
        pyplot.show()

        pyplot.plot(years[1:], rates)
        pyplot.ylabel('Growth rate (%)')
        pyplot.xlabel('Year')
        pyplot.xlim(years[1], years[-1])
        pyplot.show()
```

8.4.5

```
import matplotlib.pyplot as pyplot

def plotMeteorites():
    mFile = open('meteoritessize.txt', 'r', encoding = 'utf-8')
    header = mFile.readline()

    longitudes = []
    latitudes = []
    for line in mFile:
        row = line.split('\t')
        if row[12] != '' and row[13] != '':
         longitudes.append(float(row[12]))
         latitudes.append(float(row[13]))
    mFile.close()
    pyplot.scatter(longitudes, latitudes, 10)
    pyplot.show()
```

8.4.6

```
def plotTemps():
    tempFile = open('madison_temp.csv', 'r')
    header = tempFile.readline()

    years = []
    minTemps = []
    for line in tempFile:
        row = line.split(',')
        date = row[2]
        if date[4:6] == '01':
            years.append(date[:4])
        minTemps.append(int(row[4]) / 10)
    tempFile.close()

    pyplot.plot(range(len(minTemps)), minTemps)
    pyplot.ylabel('Minimum temperature')
    pyplot.xlabel('Year')
    pyplot.xticks(range(0, len(minTemps), 12), years)
    pyplot.xlim(0, len(minTemps))
    pyplot.show()
```

8.4.7
```
import matplotlib.pyplot as pyplot

def plotZebras():
    zebraFile = open('zebra.csv', 'r')
    header = zebraFile.readline()

    x = {}
    y = {}
    for line in zebraFile:
        row = line.split(',')
        id = row[9]
        lat = row[4]
        long = row[3]
        if (lat != '' and long != ''):
            if id not in x:
                x[id] = [float(long)]
                y[id] = [float(lat)]
            else:
                x[id].append(float(long))
                y[id].append(float(lat))
    zebraFile.close()

    colors = ['blue', 'red', 'yellow', 'green', 'brown', 'orange', 'black']
    i = 0
    for id in x:
        pyplot.scatter(x[id], y[id], 3, color = colors[i])
        i = i + 1
    pyplot.show()
```
The zebras are migrating from the Okavango Delta in northwestern Botswana southeast to the Makgadikgadi grasslands.

8.4.8
```
import matplotlib.pyplot as pyplot

def plotEd():
    edFile = open('education.txt', 'r')

    for i in range(5):
     header = edFile.readline()

    titles = edFile.readline()
    titles = titles.split('\t')

    header = edFile.readline()

    both = edFile.readline()
    bothValues = both.split('\t')
    for i in range(2, len(bothValues)):
        bothValues[i] = int(bothValues[i]) / int(bothValues[1])

    for i in range(14):
     header = edFile.readline()

    male = edFile.readline()
    maleValues = male.split('\t')
    for i in range(2, len(maleValues)):
```

```
        maleValues[i] = int(maleValues[i]) / int(maleValues[1])

    for i in range(14):
     header = edFile.readline()

    female = edFile.readline()
    femaleValues = female.split('\t')
    for i in range(2, len(femaleValues)):
        femaleValues[i] = int(femaleValues[i]) / int(femaleValues[1])

    pyplot.subplots_adjust(bottom = 0.45)          # 45% space for x tick labels
    pyplot.xticks(range(15), titles[2:], rotation = 270)   # rotated 270 degrees
    pyplot.plot(range(15), bothValues[2:], color = 'black', label = 'Both sexes')
    pyplot.plot(range(15), maleValues[2:], color = 'blue', label = 'Male')
    pyplot.plot(range(15), femaleValues[2:], color = 'red', label = 'Female')
    pyplot.legend()
    pyplot.show()
```

Section 8.5

8.5.1

```
def duplicates1(data):
    duplicateIndices = [ ]
    duplicates = [ ]
    for index in range(len(data)):
        if linearSearch(duplicateIndices, index) == -1:
            positions = linearSearchAll(data, data[index], index + 1)
            duplicateIndices.extend(positions)
            if len(positions) > 0:
                duplicates.append(data[index])
    return duplicates

def duplicates2(data):
    unique = [ ]
    duplicates = [ ]
    for item in data:
        if linearSearch(unique, item) == -1:
            unique.append(item)
        elif linearSearch(duplicates, item) == -1:
            duplicates.append(item)
    return duplicates

def duplicates3(data):
    seen = { }
    unique = [ ]
    duplicates = [ ]
    for item in data:
        if item not in seen:
            unique.append(item)
            seen[item] = True
        elif seen[item]:
            duplicates.append(item)
            seen[item] = False
    return duplicates
```

8.5.2
```
def randomList(n):
    data = []
    for i in range(n):
        data.append(random.randrange(n))
    return data

def compare(n):
    data = randomList(n)

    before = time.time()
    unique1 = removeDuplicates1(data)
    after = time.time()
    time1 = after - before

    before = time.time()
    unique2 = removeDuplicates2(data)
    after = time.time()
    time2 = after - before

    before = time.time()
    unique3 = removeDuplicates3(data)
    after = time.time()
    time3 = after - before

    print(time1, time2, time3)
```

8.5.3 The asymptotic time complexity is n^2.

8.5.4 (a)
```
def roundRobin(players):
    # initialize all players' win counts to zero
    n = len(players)
    for index1 in range(n - 1):
        for index2 in range(index1 + 1, n):
            # player1 = players[index1]
            # player2 = players[index2]
            # player1 challenges player2
            # increment the win count of the winner
    # return the player with the most wins (or tie)
```

(b) The asymptotic time complexity is n^2.

8.5.5
```
def profit(prices):
    maxProfit = 0
    bestBuy = 0
    bestSell = 0
    for buy in range(len(prices) - 1):
        for sell in range(buy + 1, len(prices)):
            if prices[sell] - prices[buy] > maxProfit:
                maxProfit = prices[sell] - prices[buy]
                bestBuy = buy
                bestSell = sell
    return (bestBuy, bestSell)
```

Section 8.6

8.6.1
```
def linearRegression(x, y):
    n = len(x) # number of points
```

```
        sumx = 0   # sum of x coordinates
        sumy = 0   # sum of y coordinates
        sumxy = 0  # sum of products of x and y coordinates
        sumxx = 0  # sum of squares of x coordinates
        for index in range(n):
            sumx = sumx + x[index]
            sumy = sumy + y[index]
            sumxy = sumxy + x[index] * y[index]
            sumxx = sumxx + x[index] * x[index]
        sumx2 = sumx ** 2   # square of sum of x coordinates

        m = (n * sumxy - sumx * sumy) / (n * sumxx - sumx2)  # slope
        b = (sumy - m * sumx) / n                            # y intercept
        return m, b
```

8.6.3

```
def readData(filename):
    dataFile = open(filename, 'r')
    line = dataFile.readline()

    HS_GPA = []
    CUM_GPA = []
    for line in dataFile:
        row = line.split(',')
        HS_GPA.append(float(row[0]))
        CUM_GPA.append(float(row[3]))
    dataFile.close()
    return HS_GPA, CUM_GPA

# linearRegression and plotRegression here

def main():
    hs, college = readData('sat.csv')
    plotRegression(hs, college, 'HS GPA', 'College GPA')

main()
```

8.6.4

```
def rSquared(x, y, m, b):
    meany = 0
    for yi in y:
        meany = meany + yi
    meany = meany / len(y)

    T = 0
    for yi in y:
        T = T + (yi - meany) ** 2

    S = 0
    for index in range(len(y)):
        xi = x[index]
        yi = y[index]
        S = S + (yi - (m * xi + b)) ** 2

    return 1 - S / T
```

8.6.5

```
def linearRegressionDeming(x, y):
    n = len(x)
```

```
    sumx = 0   # sum of x values of points
    sumy = 0   # sum of y values of points
    for index in range(n):
        sumx = sumx + x[index]
        sumy = sumy + y[index]
    meanx = sumx / n
    meany = sumy / n

    sumxy = 0
    sumxx = 0
    sumyy = 0
    for index in range(n):
     sumxx = sumxx + (x[index] - meanx) ** 2
     sumxy = sumxy + (x[index] - meanx) * (y[index] - meany)
     sumyy = sumyy + (y[index] - meany) ** 2
    sumxx = sumxx / (n - 1)
    sumxy = sumxy / (n - 1)
    sumyy = sumyy / (n - 1)

    m = (sumyy - sumxx + math.sqrt((sumyy - sumxx)**2 + 4 * sumxy**2)) / (2 * sumxy)
    b = meany - m * meanx

    return m, b
```

Section 8.7

8.7.1
```
import math

def distance(p, q):
    m = len(p)
    sumSquares = 0
    for i in range(m):
        sumSquares = sumSquares + (p[i] - q[i]) ** 2
    return math.sqrt(sumSquares)
```

8.7.2
```
def centroid(cluster, data):
    n = len(cluster)
    if n == 0:
        return random.choice(data)
    m = len(data[0])
    theCentroid = []
    for i in range(m):
        sum = 0
        for dataIndex in cluster:
            point = data[dataIndex]
            sum = sum + point[i]
        theCentroid.append(sum / n)
    return tuple(theCentroid)
```

8.7.4
```
def readFile(filename):
    inputFile = open(filename, 'r', encoding = 'utf-8')
    data = [ ]
    diagnosis = [ ]
    for line in inputFile:
        row = line.split(',')
```

```
            point = [ ]
            for value in row[1:10]:
                point.append(int(value))
            data.append(tuple(point))
            diagnosis.append(int(row[10]))
        inputFile.close()
        return data, diagnosis
```

The results are shown below. (The cluster numbers may be reversed.)

```
Cluster 0
  benign: 435 malignant: 18
Cluster 1
  benign: 9 malignant: 221
```

Cluster 0 corresponds to the program's predicted benign tumors and cluster 1 corresponds to predicted malignant tumors. The heuristic correctly classified $435/444 \approx 98\%$ of the benign tumors and $221/239 \approx 92\%$ of the malignant ones.

8.7.7
```
def kmeans(data, k):
    n = len(data)
    centroids = random.sample(data, k)

    prevClusters = None
    clusters = []
    while clusters != prevClusters:
        prevClusters = clusters[:]
        clusters = []
        for i in range(k):
            clusters.append([])

        for dataIndex in range(n):
            minIndex = 0
            for clustIndex in range(1, k):
                dist = distance(centroids[clustIndex], data[dataIndex])
                if dist < distance(centroids[minIndex], data[dataIndex]):
                    minIndex = clustIndex
            clusters[minIndex].append(dataIndex)

        for clustIndex in range(k):
            centroids[clustIndex] = centroid(clusters[clustIndex], data)

    return clusters, centroids
```

8.2 PROJECT SOLUTIONS

Project 8.1

```
import matplotlib.pyplot as pyplot

def smooth(data, window):
smoothed = []

sum = 0
for value in data[:window]:
sum = sum + value
smoothed.append(sum / window)
```

```
for index in range(1, len(data) - window + 1):
sum = sum - data[index - 1]
sum = sum + data[index + window - 1]
smoothed.append(sum / window)

return smoothed

def main():
    # Part 1: Read the deep sea isotope data

dataFile = open('2008CompilationData_Clean.csv', 'r')
header = dataFile.readline()
sites = []
ages = []
d18O_list = []
d13C_list = []
for line in dataFile:
line = line[:-1]
values = line.split(',')
site = values[0]
age = float(values[1])
d18O = values[3]
d13C = values[4]

if d18O != '' and d13C != '':  # and site in ['690', '865', '525', '527']:
sites.append(site)
ages.append(age)
d18O_list.append(float(d18O))
d13C_list.append(float(d13C))
dataFile.close()

# Plot raw d18O data.

pyplot.figure(1)
pyplot.scatter(ages, d18O_list, s = 1)
pyplot.xlim(0, 70)
pyplot.ylabel('$\delta ^{18}\mathrm{O}$')
pyplot.xlabel('ma')

# Plot raw d13C data.

pyplot.figure(2)
pyplot.scatter(ages, d13C_list, s = 1)
pyplot.xlim(0, 70)
pyplot.ylabel('$\delta ^{13}\mathrm{C}$')
pyplot.xlabel('ma')

# Part 2: plot smoothed d18O and d13C data together in subplots.

pyplot.figure(3)
pyplot.subplot(2,1,1)
pyplot.scatter(smooth(ages, 5), smooth(d18O_list, 5), s = 1)
pyplot.xlim(0, 70)
```

```
pyplot.ylim(5.5, -1)
pyplot.ylabel('$\delta ^{18}\mathrm{O}$')

pyplot.subplot(2,1,2)
pyplot.scatter(smooth(ages, 5), smooth(d13C_list, 5), s = 1, color = 'blue')
pyplot.ylim(3, -1)
pyplot.xlim(0, 70)
pyplot.ylabel('$\delta ^{13}\mathrm{C}$')
pyplot.xlabel('ma')

# Part 3: plot PETM time frame for specific sites.

pyplot.figure(4)
PETM_ages = [ages[i] for i in range(len(ages)) if (ages[i] >= 53) and (ages[i] <= 57) and
             sites[i] in ['690', '865', '527']]
PETM_d18O = [d18O_list[i] for i in range(len(ages)) if (ages[i] >= 53) and (ages[i] <= 57)
             and sites[i] in ['690', '865', '527']]
PETM_d13C = [d13C_list[i] for i in range(len(ages)) if (ages[i] >= 53) and (ages[i] <= 57)
             and sites[i] in ['690', '865', '527']]

pyplot.subplot(2, 1, 1)
pyplot.plot(smooth(PETM_ages, 5), smooth(PETM_d18O, 5), color = 'blue', label = 'd18O')
pyplot.ylabel('$\delta ^{18}\mathrm{O}$')

pyplot.subplot(2, 1, 2)
pyplot.plot(smooth(PETM_ages, 5), smooth(PETM_d13C, 5), color = 'red', label = 'd13C')
pyplot.ylabel('$\delta ^{13}\mathrm{C}$')
pyplot.xlabel('ma')

# Part 4: plot most recent 420,000 years.

dataFile = open('co2nat.txt', 'r')
header = dataFile.readline()
vostok_ages = []
vostok_CO2 = []
for line in dataFile:
line = line[:-1]
values = line.split('\t')
age = float(values[0])
CO2 = float(values[1])

vostok_ages.append(age)
vostok_CO2.append(CO2)
dataFile.close()

pyplot.figure(5)
pyplot.subplot(2, 1, 1)
pyplot.plot(smooth(vostok_ages, 5), smooth(vostok_CO2, 5))
pyplot.ylabel('$\mathrm{CO}_2 \mathrm{(ppmv)}$')
pyplot.xlabel('years ago')

recent_ages = [ages[i] for i in range(len(ages)) if (ages[i] <= 0.42) and sites[i]
               in ['607', '659', '849']]
recent_d18O = [d18O_list[i] for i in range(len(ages)) if (ages[i] <= 0.42) and sites[i]
```

```
               in ['607', '659', '849']]
pyplot.subplot(2, 1, 2)
pyplot.plot(smooth(recent_ages, 5), smooth(recent_d18O, 5))
pyplot.ylabel('$\delta ^{18}\mathrm{O}$')
pyplot.xlabel('ma')

# Part 5: plot Keeling data

dataFile = open('weekly_mlo.csv', 'r')
line = dataFile.readline()
while line[0] in '"':
line = dataFile.readline()
# value of line here should be '%'

keeling_years = []
keeling_CO2 = []
for line in dataFile:
line = line[:-1]
values = line.split(',')
date = values[0]
year = float(date[:4]) + (float(date[5:7]) + float(date[8:10]) / 31) / 12
CO2 = float(values[1])

keeling_years.append(year)
keeling_CO2.append(CO2)
dataFile.close()

pyplot.figure(6)
pyplot.plot(keeling_years, keeling_CO2)
pyplot.ylabel('$\mathrm{CO}_2 \mathrm{(ppmv)}$')
pyplot.xlabel('year')

pyplot.show()

main()
```

Project 8.2

```
#!/usr/bin/env python3.3

import matplotlib.pyplot as pyplot

def fix(rate):
    return round(1000 * rate) / 10

def main():
    # Part 1

    fileName = 'ACS_12_1YR_B23006_with_ann.txt'
    dataFile = open(fileName, 'r', encoding = 'utf-8')
    header1 = dataFile.readline()
    header2 = dataFile.readline()

    names = []
```

```
totals = []
ratesNoHS = []
ratesHS = []
ratesSomeCollege = []
ratesCollegeGrad = []
for line in dataFile:
    values = line.split('\t')

    names.append(values[2])
    totals.append(int(values[3]))
    ratesNoHS.append(fix(int(values[15]) / int(values[11])))
    ratesHS.append(fix(int(values[29]) / int(values[25])))
    ratesSomeCollege.append(fix(int(values[43]) / int(values[39])))
    ratesCollegeGrad.append(fix(int(values[57]) / int(values[53])))

dataFile.close()

pyplot.plot(range(len(totals)), ratesNoHS, label = 'No HS diploma')
pyplot.plot(range(len(totals)), ratesHS, label = 'HS graduates')
pyplot.plot(range(len(totals)), ratesSomeCollege, label = 'Some college')
pyplot.plot(range(len(totals)), ratesCollegeGrad, label = 'College graduates')
pyplot.xticks(range(len(totals)), names, rotation = 270, fontsize = 'small')
pyplot.ylabel('Unemployment rate (%)')
pyplot.legend()
pyplot.show()

# Part 2

totalsSorted = totals[:]
totalsSorted.sort()
cutoff = totalsSorted[-30]

names30 = []
totals30 = []
ratesNoHS30 = []
ratesHS30 = []
ratesSomeCollege30 = []
ratesCollegeGrad30 = []
for index in range(len(totals)):
    if totals[index] >= cutoff:
        names30.append(names[index])
        totals30.append(totals[index])
        ratesNoHS30.append(ratesNoHS[index])
        ratesHS30.append(ratesHS[index])
        ratesSomeCollege30.append(ratesSomeCollege[index])
        ratesCollegeGrad30.append(ratesCollegeGrad[index])

pyplot.plot(range(30), ratesNoHS30, label = 'No HS diploma')
pyplot.plot(range(30), ratesHS30, label = 'HS graduates')
pyplot.plot(range(30), ratesSomeCollege30, label = 'Some college')
pyplot.plot(range(30), ratesCollegeGrad30, label = 'College graduates')
pyplot.xticks(range(30), names30, rotation = 270, fontsize = 'small')
pyplot.ylabel('Unemployment rate (%)')
pyplot.legend()
```

```
    pyplot.show()

    # Part 3

    print('Areas with inverted unemployment rates:')
    for index in range(30):
        if ratesNoHS30[index] < ratesHS30[index]:
            print(names[index])

    mins = [0, 0, 0, 0]
    maxs = [0, 0, 0, 0]
    maxDiffIndex = 0
    for index in range(1, 30):
        if ratesNoHS30[index] < ratesNoHS30[mins[0]]:
            mins[0] = index
        if ratesNoHS30[index] > ratesNoHS30[maxs[0]]:
            maxs[0] = index
        if ratesHS30[index] < ratesHS30[mins[1]]:
            mins[1] = index
        if ratesHS30[index] > ratesHS30[maxs[1]]:
            maxs[1] = index
        if ratesSomeCollege30[index] < ratesSomeCollege30[mins[2]]:
            mins[2] = index
        if ratesSomeCollege30[index] > ratesSomeCollege30[maxs[2]]:
            maxs[2] = index
        if ratesCollegeGrad30[index] < ratesCollegeGrad30[mins[3]]:
            mins[3] = index
        if ratesCollegeGrad30[index] > ratesCollegeGrad30[maxs[3]]:
            maxs[3] = index
        if ratesHS30[index] - ratesCollegeGrad30[index] > ratesHS30[maxDiffIndex] -
           ratesCollegeGrad30[maxDiffIndex]:
            maxDiffIndex = index

    print('No high school diploma:', names[mins[0]], '-', names[maxs[0]])
    print('High school graduate:', names[mins[1]], '-', names[maxs[1]])
    print('Some college:', names[mins[2]], '-', names[maxs[2]])
    print('College graduate:', names[mins[3]], '-', names[maxs[3]])
    print('Maximum difference:', names[maxDiffIndex],
          ratesHS30[maxDiffIndex] - ratesCollegeGrad30[maxDiffIndex])

    collegeDict = {}
    for index in range(30):
        collegeDict[ratesCollegeGrad30[index]] = names[index]

    rates = list(collegeDict.keys())
    rates.sort()

    for rate in rates:
        print('{0:<45} {1:>4.1f}%'.format(collegeDict[rate], rate))

main()
```

Project 8.3

```
import random
import matplotlib.pyplot as pyplot

NUM_CUSTOMERS = 1000

def linearRegression(x, y):
    n = len(x) # number of points
    sumx = 0   # sum of x coordinates
    sumy = 0   # sum of y coordinates
    sumxy = 0  # sum of products of x and y coordinates
    sumxx = 0  # sum of squares of x coordinates
    for index in range(n):
        sumx = sumx + x[index]
        sumy = sumy + y[index]
        sumxy = sumxy + x[index] * y[index]
        sumxx = sumxx + x[index] * x[index]
    sumx2 = sumx ** 2   # square of sum of x coordinates

    m = (n * sumxy - sumx * sumy) / (n * sumxx - sumx2)  # slope
    b = (sumy - m * sumx) / n                            # y intercept
    return m, b

def randCustomers(n):
    customers = []
    for index in range(n):
        customers.append(random.gauss(4, 1.5))
        if customers[-1] < 0:
            customers[-1] = 0
    return customers

def histPrices(customers):
    pyplot.hist(customers)
    pyplot.xlabel('Maximum price willing to pay')
    pyplot.ylabel('Number of customers')
    pyplot.show()

def sales(customers, price):
    count = 0
    for cust in customers:
        if cust >= price:
            count = count + 1
    return count

def plotDemand(customers, lowPrice, highPrice, step):
    salesList = []
    priceList = []
    price = lowPrice
    while price <= highPrice:
        salesList.append(sales(customers, price))
        priceList.append(price)
        price = price + step
    pyplot.scatter(priceList, salesList)
```

```
    m, b = linearRegression(priceList, salesList)
    pyplot.plot([lowPrice, highPrice], [m * lowPrice + b, m * highPrice + b], color = 'red')
    pyplot.title('Demand Curve')
    pyplot.xlabel('Selling price')
    pyplot.ylabel('Number of sales')
    pyplot.show()

def profits(customers, lowPrice, highPrice, step, perCost, fixedCost):
    profitList = []
    priceList = []
    maxProfit = float('-inf')
    maxPrice = lowPrice
    price = lowPrice
    while price <= highPrice:
        sold = sales(customers, price)
        cost = perCost * sold + fixedCost
        profit = sold * price - cost
        if profit > maxProfit:
            maxProfit = profit
            maxPrice = price
            maxSold = sold
        profitList.append(profit)
        priceList.append(price)
        price = price + step

    pyplot.plot(priceList, profitList)
    pyplot.xlabel('Selling price')
    pyplot.ylabel('Profit')
    pyplot.show()

    return maxProfit, maxPrice, maxSold

def main():
    customers = randCustomers(NUM_CUSTOMERS)
    histPrices(customers)
    plotDemand(customers, 0, 8, 0.25)
    maxProfit, maxPrice, maxSold = profits(customers, 0, 8, 0.25,
                                           10 * (8 / 454) + 0.05 * 0.5, 500)
    print(maxProfit, maxPrice, maxSold)

main()
```

Project 8.4

```
import matplotlib.pyplot as pyplot

def readData(filename):
    dataFile = open(filename, 'r')
    line = dataFile.readline()

    HS_GPA = []
    SAT_M = []
    SAT_V = []
    CUM_GPA = []
```

```
    for line in dataFile:
        row = line.split(',')
        HS_GPA.append(float(row[0]))
        SAT_M.append(int(row[1]))
        SAT_V.append(int(row[2]))
        CUM_GPA.append(float(row[3]))
    dataFile.close()
    return HS_GPA, SAT_M, SAT_V, CUM_GPA

def plotData(HS_GPA, SAT_M, SAT_V, CUM_GPA):
    x = range(len(HS_GPA))
    pyplot.figure()
    pyplot.subplot(4, 1, 1)
    pyplot.scatter(x, HS_GPA)
    pyplot.ylabel('High school GPA')
    pyplot.xlim(0, len(HS_GPA))

    pyplot.subplot(4, 1, 2)
    pyplot.scatter(x, SAT_M)
    pyplot.ylabel('Math SAT')
    pyplot.xlim(0, len(HS_GPA))

    pyplot.subplot(4, 1, 3)
    pyplot.scatter(x, SAT_V)
    pyplot.ylabel('Verbal SAT')
    pyplot.xlim(0, len(HS_GPA))

    pyplot.subplot(4, 1, 4)
    pyplot.scatter(x, CUM_GPA)
    pyplot.ylabel('College GPA')
    pyplot.xlabel('Student')
    pyplot.xlim(0, len(HS_GPA))

    pyplot.show()

def linearRegression(x, y):
    n = len(x) # number of points
    sumx = 0   # sum of x coordinates
    sumy = 0   # sum of y coordinates
    sumxy = 0  # sum of products of x and y coordinates
    sumxx = 0  # sum of squares of x coordinates
    for index in range(n):
        sumx = sumx + x[index]
        sumy = sumy + y[index]
        sumxy = sumxy + x[index] * y[index]
        sumxx = sumxx + x[index] * x[index]
    sumx2 = sumx ** 2   # square of sum of x coordinates

    m = (n * sumxy - sumx * sumy) / (n * sumxx - sumx2)  # slope
    b = (sumy - m * sumx) / n                            # y intercept
    return m, b

def rSquared(x, y, m, b):
    meany = 0
```

```
    for yi in y:
        meany = meany + yi
    meany = meany / len(y)

    T = 0
    for yi in y:
        T = T + (yi - meany) ** 2

    S = 0
    for index in range(len(y)):
        xi = x[index]
        yi = y[index]
        S = S + (yi - (m * xi + b)) ** 2

    return 1 - S / T

def plotRegression(x, y, xLabel, yLabel):
    """Plot points in x and y with a linear regression line.

    Parameters:
        x: a list of x values (independent variable)
        y: a list of y values (dependent variable)
        xLabel: a string to label the x axis
        yLabel: a string to label the y axis

    Return value: None
    """

    pyplot.scatter(x, y)
    m, b = linearRegression(x, y)
    minX = min(x)
    maxX = max(x)
    pyplot.plot([minX, maxX], [m * minX + b, m * maxX + b], color = 'red')
    pyplot.xlabel(xLabel)
    pyplot.ylabel(yLabel)
    pyplot.show()

    return rSquared(x, y, m, b)

def main():
    HS_GPA, SAT_M, SAT_V, CUM_GPA = readData('sat.csv')
    plotData(HS_GPA, SAT_M, SAT_V, CUM_GPA)

    SATcombined = []
    for index in range(len(SAT_M)):
        SATcombined.append(SAT_M[index] + SAT_V[index])

    r = plotRegression(HS_GPA, CUM_GPA, 'High school GPA', 'College GPA')
    print('R-squared for high school GPA is', r)
    r = plotRegression(SAT_M, CUM_GPA, 'SAT math', 'College GPA')
    print('R-squared for SAT math is', r)
    r = plotRegression(SAT_V, CUM_GPA, 'SAT verbal', 'College GPA')
    print('R-squared for SAT verbal is', r)
    r = plotRegression(SATcombined, CUM_GPA, 'SAT combined', 'College GPA')
```

```
    print('R-squared for SAT combined is', r)

main()
```

Project 8.5

```
import matplotlib.pyplot as pyplot
import math

def linearRegression(x, y):
    n = len(x) # number of points
    sumx = 0   # sum of x coordinates
    sumy = 0   # sum of y coordinates
    sumxy = 0  # sum of products of x and y coordinates
    sumxx = 0  # sum of squares of x coordinates
    for index in range(n):
        sumx = sumx + x[index]
        sumy = sumy + y[index]
        sumxy = sumxy + x[index] * y[index]
        sumxx = sumxx + x[index] * x[index]
    sumx2 = sumx ** 2   # square of sum of x coordinates

    m = (n * sumxy - sumx * sumy) / (n * sumxx - sumx2)  # slope
    b = (sumy - m * sumx) / n                            # y intercept
    return m, b

def readData(filename):
    dataFile = open(filename, 'r')
    line = dataFile.readline()
    while line[0] == '#':
        line = dataFile.readline()
    headerLine = line
    columnDefLine = dataFile.readline()

    flows = []
    heights = []
    for line in dataFile:
        row = line.split('\t')
        flow = row[4]
        height = row[6]
        if (flow != ''):
            flows.append(float(flow))
        else:
            flows.append(0)
        if (height != ''):
            heights.append(float(height))
        else:
            heights.append(0)
            print(flow)
    dataFile.close()
    return flows, heights

def getRecurrenceIntervals(n):
    recurrenceIntervals = []
```

```
    for rank in range(n, 0, -1):
        r = (n + 1) / rank
        recurrenceIntervals.append(r)
    return recurrenceIntervals

def plotRecurrenceIntervals(heights):
    heightsSorted = []  # make a copy of heights without zeros
    for height in heights:
        if height > 0:
            heightsSorted.append(height)
    heightsSorted.sort()        # sort the copy
    recIntervals = getRecurrenceIntervals(len(heightsSorted))
    pyplot.scatter(recIntervals, heightsSorted)
    pyplot.xlabel('Recurrence Interval')
    pyplot.ylabel('Gauge Height (ft)')
    pyplot.show()

def plotLogRecurrenceIntervals(heights):
    heightsSorted = []
    for height in heights:  # make a copy of heights without any zeros
        if height > 0:      # (could use a list comprehension)
            heightsSorted.append(height)
    heightsSorted.sort()    # sort the copy
    recIntervals = getRecurrenceIntervals(len(heightsSorted))

    logRecIntervals = []
    for r in recIntervals:
        logRecIntervals.append(math.log(r, 10))

    pyplot.scatter(logRecIntervals, heightsSorted)

    m, b = linearRegression(logRecIntervals, heightsSorted)
    pyplot.plot([0, 2], [m * 0 + b, m * 2 + b], color = 'red')
    pyplot.xticks([0, 1, 1.477, 2], ['1', '10', '30', '100'])

    pyplot.xlabel('Recurrence Interval')
    pyplot.ylabel('Gauge Height (ft)')
    pyplot.show()

    return m * 2 + b

def plotFlowsHeights(flows, heights):
    flows = [flows[i] for i in range(len(flows)) if heights[i] > 0]
    heights = [heights[i] for i in range(len(heights)) if heights[i] > 0]
    pyplot.scatter(flows, heights)
    m, b = linearRegression(flows, heights)
    minf = min(flows)                       # min x value
    maxf = max(flows)                       # max x value
    pyplot.plot([minf, maxf], \
             [m * minf + b, m * maxf + b], \
             color = 'red')                 # y = m * x + b
    pyplot.xlabel('Peak Discharge (cfs)')
    pyplot.ylabel('Gauge Height (ft)')
    pyplot.show()
```

```
def plotLogRecurrenceIntervals2(flows):
    flowsSorted = flows[:]
    flowsSorted.sort()    # sort the copy
    recIntervals = getRecurrenceIntervals(len(flowsSorted))

    logRecIntervals = []
    for r in recIntervals:
        logRecIntervals.append(math.log(r, 10))

    pyplot.scatter(logRecIntervals, flowsSorted)

    m, b = linearRegression(logRecIntervals, flowsSorted)
    pyplot.plot([0, 2], [m * 0 + b, m * 2 + b], color = 'red')
    pyplot.xticks([0, 1, 1.477, 2], ['1', '10', '30', '100'])

    pyplot.xlabel('Recurrence Interval')
    pyplot.ylabel('Peak streamflow (cfs)')
    pyplot.show()

    return m * 2 + b

def main():
    flows, heights = readData('snake_peak.txt')
    plotRecurrenceIntervals(heights)
    height100 = plotLogRecurrenceIntervals(heights)
    plotFlowsHeights(flows, heights)
    flow100 = plotLogRecurrenceIntervals2(flows)
    m, b = linearRegression(flows, heights)
    height100_2 = m * flow100 + b

    print(height100, height100_2)

main()
```

Project 8.6

```
def readVotes(fileName):
    voteFile = open(fileName, 'r', encoding = 'utf-8')
    ballots = []
    for line in voteFile:
        line = line.strip()
        ballot = line.split()
        ballots.append(ballot)
    voteFile.close()

    return ballots

def printWinners(points):
    winner = None
    maxPoints = 0
    for candidate in points:
        if points[candidate] > maxPoints:
            winner = candidate
```

```
            maxPoints = points[candidate]

    winners = []
    for candidate in points:
        if points[candidate] == points[winner]:
            winners.append(candidate)

    if len(winners) > 1:
        print('There was a tie among the following candidates:', winners)
    else:
       print('The winner is', winners[0])

# PLURALITY

def plurality(ballots):
    votes = {}
    for ballot in ballots:
        first = ballot[0]
        if first in votes:
            votes[first] = votes[first] + 1
        else:
            votes[first] = 1

    printWinners(votes)

# BORDA

def processBallot(points, ballot):
    value = len(ballot) - 1
    for candidate in ballot:
        if candidate in points:
            points[candidate] = points[candidate] + value
        else:
            points[candidate] = value
        value = value - 1

def borda(ballots):
    points = {}
    for ballot in ballots:
        processBallot(points, ballot)

    printWinners(points)

# CONDORCET

def head2head(ranks, candidate1, candidate2):
    votes1 = 0
    votes2 = 0
    for ranking in ranks:
        if ranking[candidate1] < ranking[candidate2]:
            votes1 = votes1 + 1
        else:
            votes2 = votes2 + 1
```

```
    if votes1 > votes2:
        return candidate1
    elif votes2 > votes1:
        return candidate2
    else:
        return None

def condorcet(ballots):
    candidates = ballots[0]
    wins = {}
    for candidate in candidates:
        wins[candidate] = 0

    ranks = []
    for ballot in ballots:
        ranking = {}
        for index in range(len(ballot)):
            ranking[ballot[index]] = index
        ranks.append(ranking)

    for candidate1 in candidates:
        for candidate2 in candidates:
            if candidate2 != candidate1:
                winner = head2head(ranks, candidate1, candidate2)
                if winner == candidate1:
                    wins[candidate1] = wins[candidate1] + 1
    winner = None
    for candidate in candidates:
        if wins[candidate] == len(candidates) - 1:
            winner = candidate

    if winner != None:
        print('The winner is', winner)
    else:
        print('There is no Condorcet winner.')

def main():
    ballots = readVotes('votes1.txt')
    plurality(ballots)
    borda(ballots)
    condorcet(ballots)

main()
```

CHAPTER 9

Flatland

9.1 EXERCISE SOLUTIONS

Section 9.1

9.1.1
```
scores = [[10305, 700, 610], [11304, 680, 590], [10254, 710, 730],
          [12007, 650, 690], [10089, 780, 760]]
```

9.1.2 (a) `scores[4][1]`

(b) `scores[1][2]`

(c) `scores[0][1]`

(d) `scores[3][2]`

9.1.3 (a)
```
scores = [10305, 11304, 10254, 12007, 10089], [700, 680, 710, 650, 780],
          [610, 590, 730, 690, 760]]
```

(b) i. `scores[1][4]`

ii. `scores[2][1]`

iii. `scores[1][0]`

iv. `scores[2][3]`

(c) When we iterate over a file, we get the data row by row, making it easy to insert into a list of rows. If you stored the data as a list of columns instead, we would need to iterate over the entire file once for every column.

9.1.4
```
def queryTemps():
    temps = readData()
    key = input('Minimum temperature for which date (q to quit)? ')
    while key != 'q':
        temp = getMinTemp(temps, key)
        if temp != None:
            print('The minimum temperature for', key, \
                      'was', temp / 10, 'degrees Celsius.')
        else:
            print('That date was was not found.')
        print()
        key = input('Minimum temperature for which date (q to quit)? ')
```

9.1.5
```
def getMaxTemp(table, date):
    for r in range(len(table)):
        if table[r][0] == date:
            return table[r][1]
    return None
```

9.1.6
```
def readData():
    dataFile = open('madison_temp.csv', 'r')
    header = dataFile.readline()
    table = {}
    for line in dataFile:
        row = line.split(',')
        row[3] = int(row[3])
        row[4] = int(row[4])
        table[row[2]] = row[3:]
    dataFile.close()
    return table

def getMinTemp(table, date):
    if date in table:
        return table[date][1]
    return None
```

9.1.7
```
import urllib.request as web

def readQuakes():
    url = 'http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_month.csv'
    quakeFile = web.urlopen(url)
    header = quakeFile.readline()

    table = []
    for line in quakeFile:
        line = line.decode('utf-8')
        row = line.split(',')
        latitude = float(row[1])
        longitude = float(row[2])
        depth = float(row[3])
        magnitude = float(row[4])
        table.append([latitude, longitude, depth, magnitude])
    quakeFile.close()

    return table
```

9.1.8
```
def printTable(quakesTable):
    print('Latitude  Longitude  Depth  Magnitude')
    print('--------  ---------  -----  ---------')
    for row in quakesTable:
        print('{0:>8.2f}  {1:>9.2f}  {2:>5.1f}  {3:>6.1f}'.format(row[0], row[1],
              row[2], row[3]))
```

9.1.9
```
def queryQuakes(quakesTable):
    magnitude = input('Minimum magnitude (q to quit)? ')
    while magnitude != 'q':
        magnitude = float(magnitude)
        queryTable = []
        for row in quakesTable:
            if row[3] >= magnitude:
                queryTable.append(row)
        if queryTable == []:
            print('There were no earthquakes with magnitude at least',
                  str(magnitude) + '.')
        else:
```

```
            print()
            printTable(queryTable)
        print()
        magnitude = input('Minimum magnitude (q to quit)? ')
```

Section 9.2

9.2.2
```
def neighborhood(grid, row, column):
    offsets = [(-1, -1), (-1, 0), (-1, 1), (0, -1),
               (0, 1), (1, -1), (1, 0), (1, 1)]
    count = 0
    for offset in offsets:
        r = (row + offset[0]) % ROWS
        c = (column + offset[1]) % COLUMNS
        if grid[r][c] == ALIVE:
            count = count + 1
    return count
```

9.2.3
```
def printGrid(grid):
    rows = len(grid)
    columns = len(grid[0])

    for r in range(rows):
        for c in range(columns):
            print(grid[r][c], end = ' ')
        print()
```

9.2.4
```
def multiplicationTable(n):
    table = []
    for r in range(n):
        row = []
        for c in range(n):
            row.append(r * c)
        table.append(row)
    return table
```

9.2.5
```
def diagonal(n):
    grid = []
    for r in range(n):
        row = []
        for c in range(n):
            if c == r:
                row.append(1)
            else:
                row.append(0)
        grid.append(row)
    return grid
```

9.2.6
```
def diagonal2(n):
    grid = []
    for r in range(n):
        row = []
        for c in range(n):
            if c <= r:
                row.append(1)
            else:
                row.append(0)
```

```
            grid.append(row)
        return grid
```

9.2.7
```
def sums(grid):
    sumAll = 0
    for row in range(len(grid)):
        sum = 0
        for col in range(len(grid[0])):
            sum = sum + grid[row][col]
        print('Row', row, 'sum =', sum)
        sumAll = sumAll + sum
    for col in range(len(grid[0])):
        sum = 0
        for row in range(len(grid)):
            sum = sum + grid[row][col]
        print('Column', col, 'sum =', sum)
    print('Total sum =', sumAll)
```

9.2.8
```
def find(grid, target):
    for row in range(len(grid)):
        for col in range(len(grid[0])):
            if grid[row][col] == target:
                return (row, col)
    return (-1, -1)
```

9.2.9
```
def checkerboard():
    board = []
    for r in range(8):
        row = []
        for c in range(8):
            if (r + c) % 2 == 1:
                row.append('B')
            else:
                row.append('W')
        board.append(row)
    return board
```

9.2.10
```
import random

def magic(n):  # n is odd
    square = []
    for r in range(n):
        square.append([0] * n)

    row = random.randrange(n)
    col = random.randrange(n)
    for number in range(1, n * n + 1):
        square[row][col] = number
        nextRow = (row + 1) % n
        nextCol = (col + 1) % n
        if square[nextRow][nextCol] != 0:
            row = (row - 1) % n
        else:
            row = nextRow
            col = nextCol
    return square
```

9.2.11
```
def emptyGrid(rows, columns):
```

```
    grid = { }
    for r in range(rows):
        for c in range(columns):
            grid[(r,c)] = DEAD
    return grid

def initialize(grid, coordinates, tortoise):
    for (r, c) in coordinates:
        grid[(r,c)] = ALIVE
        drawSquare((r, c), 'black', tortoise)

def neighborhood(grid, row, column):
    offsets = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]
    count = 0
    for offset in offsets:
        r = row + offset[0]
        c = column + offset[1]
        if (r,c) in grid:
            if grid[(r,c)] == ALIVE:
                count = count + 1
    return count

def life(rows, columns, generations, initialCells, tortoise):
    grid = emptyGrid(rows, columns)
    drawGrid(rows, columns, tortoise)
    initialize(grid, initialCells, tortoise)
    for g in range(generations):
        newGrid = copy.deepcopy(grid)
        for r in range(rows):
            for c in range(columns):
                neighbors = neighborhood(grid, r, c)
                if grid[(r,c)] == ALIVE and (neighbors < 2 or neighbors > 3):
                    newGrid[(r,c)] = DEAD
                    drawSquare((r, c), 'white', tortoise)
                elif grid[(r,c)] == DEAD and neighbors == 3:  # rule 4
                    newGrid[(r,c)] = ALIVE
                    drawSquare((r, c), 'black', tortoise)
        grid = newGrid
    return grid
```

Section 9.3

9.3.1
```
def color2gray(color):
    luminance = int(0.2126 * color[0] + 0.7152 * color[1] + 0.0722 * color[2])
    return (luminance, luminance, luminance)
```

9.3.2
```
def warmPixel(color, factor):
    red = min(255, int((1 + factor) * color[0]))
    green = min(255, int((1 + factor) * color[1]))
    blue = color[2]
    return (red, green, blue)

def warm(photo, factor):
    width = photo.width()
```

```
        height = photo.height()
        newPhoto = image.Image(width, height, title = 'Warm image')
        for y in range(height):
            for x in range(width):
                color = photo.get(x, y)
                newPhoto.set(x, y, warmPixel(color, factor))
        return newPhoto
```

9.3.3

```
    def coolPixel(color, factor):
        red = color[0]
        green = color[1]
        blue = min(255, int((1 + factor) * color[2]))
        return (red, green, blue)

    def cool(photo, factor):
        width = photo.width()
        height = photo.height()
        newPhoto = image.Image(width, height, title = 'Cool image')
        for y in range(height):
            for x in range(width):
                color = photo.get(x, y)
                newPhoto.set(x, y, coolPixel(color, factor))
        return newPhoto
```

9.3.4

```
    def luminancePixel(color, factor):
        red = min(255, int((1 + factor) * color[0]))
        green = min(255, int((1 + factor) * color[1]))
        blue = min(255, int((1 + factor) * color[2]))
        return (red, green, blue)

    def luminance(photo, factor):
        width = photo.width()
        height = photo.height()
        newPhoto = image.Image(width, height, title = 'Brightened image')
        for y in range(height):
            for x in range(width):
                color = photo.get(x, y)
                newPhoto.set(x, y, luminancePixel(color, factor))
        return newPhoto
```

9.3.5

```
    def negativePixel(color):
        return (255 - color[0], 255 - color[1], 255 - color[2])

    def negative(photo):
        width = photo.width()
        height = photo.height()
        newPhoto = image.Image(width, height, title = 'Negative image')
        for y in range(height):
            for x in range(width):
                color = photo.get(x, y)
                newPhoto.set(x, y, negativePixel(color))
        return newPhoto
```

9.3.6

```
    def flipHorzontal(photo):
        width = photo.width()
        height = photo.height()
        newPhoto = image.Image(width, height, title = 'Flipped image')
```

```
        for y in range(height):
            for x in range(width):
                color = photo.get(x, y)
                newPhoto.set(width - x - 1, y, color)
        return newPhoto
```

9.3.7
```
def mirror(photo):
    width = photo.width()
    height = photo.height()
    newPhoto = image.Image(width, height, title = 'Mirrored image')
    for y in range(height):
        for x in range(width // 2):
            color = photo.get(x, y)
            newPhoto.set(x, y, color)
            newPhoto.set(width - x - 1, y, color)
    return newPhoto
```

9.3.8
```
def reducePixels(photo, x, y):
    offsets = [(0, 0), (0, 1), (1, 0), (1, 1)]
    red = 0
    green = 0
    blue = 0
    for offset in offsets:
        color = photo.get(x + offset[0], y + offset[1])
        red = red + color[0]
        green = green + color[1]
        blue = blue + color[2]
    return (red // 4, green // 4, blue // 4)

def reduce(photo):
    width = photo.width()
    height = photo.height()
    newPhoto = image.Image(width // 2, height // 2, title = 'Reduced image')
    for y in range(0, height, 2):
        for x in range(0, width, 2):
            color = reducePixels(photo, x, y)
            newPhoto.set(x // 2, y // 2, color)
    return newPhoto
```

9.3.9
```
def blurPixel(photo, x, y):
    width = photo.width()
    height = photo.height()
    offsets = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1),
               (1, 0), (1, 1)]
    neighbors = 0
    red = 0
    green = 0
    blue = 0
    for offset in offsets:
        if x + offset[0] >= 0 and x + offset[0] < width and y + offset[1] >= 0 and
           y + offset[1] < height:
            neighbors = neighbors + 1
            color = photo.get(x + offset[0], y + offset[1])
            red = red + color[0]
            green = green + color[1]
            blue = blue + color[2]
```

```
        return (red // neighbors, green // neighbors, blue // neighbors)

    def blur(photo):
        width = photo.width()
        height = photo.height()
        newPhoto = image.Image(width, height, title = 'Blurred image')
        for y in range(height):
            for x in range(width):
                newPhoto.set(x, y, blurPixel(photo, x, y))
        return newPhoto
```

9.3.10

```
def blur(photo, times):
    if times <= 0:
        return photo
    newPhoto = blur1(photo)
    for count in range(times - 1):
        newPhoto = blur1(newPhoto)
    return newPhoto
```

9.3.11

```
def crop(photo, x1, y1, x2, y2):
    width = photo.width()
    height = photo.height()
    if x1 < 0 or x1 >= width or x2 <= x1 or x2 >= width:
        return None
    if y1 < 0 or y1 >= height or y2 <= y1 or y2 >= height:
        return None
    newPhoto = image.Image(x2 - x1 + 1, y2 - y1 + 1, title = 'Cropped image')
    for y in range(y1, y2 + 1):
        for x in range(x1, x2 + 1):
            color = photo.get(x, y)
            newPhoto.set(x - x1, y - y1, color)
    return newPhoto
```

9.2 PROJECT SOLUTIONS

Project 9.1

```
import turtle
import random
import copy

SCALE = 10

PROP_VACANT = 0.10
PROP_YELLOW = 0.45
PROP_GREEN = 0.45
TOLERANCE = 0.375

GREEN = -1   # Star-Belly Sneetch
YELLOW = 1   # Plain-Belly Sneetch
VACANT = 0

def drawSquare(pos, color, tortoise):
    """Draws one square in the given color at the
       given position in the grid.
```

```
    Parameters:
        pos: a (row, column) tuple
        color: a color string
        tortoise: a Turtle object

    Return value: None
    """

    (row, column) = pos
    screen = tortoise.getscreen()
    rows = int(screen.canvheight / screen.yscale)
    row = rows - row - 1
    tortoise.shape(color)
    tortoise.up()
    tortoise.goto(column, row + 1)
    tortoise.stamp()

def drawGrid(rows, columns, tortoise):
    """Draws an empty grid using turtle graphics.

    Parameters:
        rows: the number of rows in the grid
        columns: the number of columns in the grid
        tortoise: a Turtle object

    Return value: None
    """

    tortoise.pencolor('gray')
    for row in range(rows + 1):
        tortoise.up()
        tortoise.goto(0, row)
        tortoise.down()
        tortoise.goto(columns, row)
    for column in range(columns + 1):
        tortoise.up()
        tortoise.goto(column, 0)
        tortoise.down()
        tortoise.goto(column, rows)

def createSquares(screen, colors):
    """Creates square shapes in the given colors to be used
       as turtle graphics stamps in a grid.

    Parameters:
        screen: a Screen object
        colors: a list of color strings

    Return value: None
    """

    square = ((0, 0), (0, SCALE), (SCALE, SCALE), (SCALE, 0))
    for color in colors:
```

```
        squareShape = turtle.Shape('compound')
        squareShape.addcomponent(square, color, 'gray')
        screen.register_shape(color, squareShape)

def neighborhood(grid, row, column):
    neighbors = [(-1, -1), (-1, 0), (-1, 1), (1, -1), (1, 0), (1, 1), (0, -1), (0, 1)]
    rows = len(grid)
    columns = len(grid[0])
    me = grid[row][column]
    other = 0
    like = 0
    for n in neighbors:
        c = column + n[0]
        r = row + n[1]
        if c in range(columns) and r in range(rows):
            if grid[r][c] == -me:
                other = other + 1
            elif grid[r][c] != 0:
                like = like + 1
    if other + like > 0:
        return other / (other + like)
    else:
        return 0

def schelling(rows, columns, passes, tortoise):
    drawGrid(rows, columns, tortoise)
    grid = []
    vacancies = []
    for r in range(rows):
        row = []
        for c in range(columns):
            p = random.random()
            if p < PROP_VACANT:
                row.append(VACANT)
                vacancies.append((r, c))
            elif p < PROP_VACANT + PROP_YELLOW:
                row.append(YELLOW)
                drawSquare((r, c), "yellow", tortoise)
            else:
                row.append(GREEN)
                drawSquare((r, c), "darkgreen", tortoise)
        grid.append(row)

    for p in range(passes):
        newGrid = copy.deepcopy(grid)
        for r in range(rows):
            for c in range(columns):
                if grid[r][c] != VACANT and neighborhood(grid, r, c) > TOLERANCE:
                    new = random.choice(vacancies)
                    vacancies.remove(new)
                    newGrid[new[0]][new[1]] = grid[r][c]

                    newGrid[r][c] = VACANT
                    vacancies.append((r, c))
```

```
                    if newGrid[new[0]][new[1]] == YELLOW:
                        drawSquare(new, "yellow", tortoise)
                    else:
                        drawSquare(new, "darkgreen", tortoise)
                    drawSquare((r, c), "white", tortoise)
        grid = newGrid

def main():
    rows = 20
    columns = 20
    tortoise = turtle.Turtle()
    screen = tortoise.getscreen()
    screen.setup(columns * SCALE + 20, rows * SCALE + 20)
    screen.setworldcoordinates(0, 0, columns, rows)
    screen.tracer(100)
    tortoise.hideturtle()
    createSquares(screen, ['darkgreen', 'yellow', 'white'])

    schelling(rows, columns, 100, tortoise)

    screen.update()
    screen.exitonclick()

main()
```

Project 9.2

```
import turtle
import random
import math

# Assume a square grid.

PROP_UP = 0.5
T = 0.5
SCALE = 20

def drawSquare(pos, color, tortoise):
    (row, column) = pos
    screen = tortoise.getscreen()
    rows = int(screen.canvheight / screen.yscale)
    row = rows - row - 1
    tortoise.shape(color)
    tortoise.up()
    tortoise.goto(column, row + 1)
    tortoise.stamp()

def drawGrid(rows, columns, tortoise):
    tortoise.pencolor('gray')
    for row in range(rows + 1):
        tortoise.up()
        tortoise.goto(0, row)
        tortoise.down()
```

```
        tortoise.goto(columns, row)
    for column in range(columns + 1):
        tortoise.up()
        tortoise.goto(column, 0)
        tortoise.down()
        tortoise.goto(column, rows)

def createSquares(screen, colors):
    square = ((0, 0), (0, SCALE), (SCALE, SCALE), (SCALE, 0))
    for color in colors:
        squareShape = turtle.Shape('compound')
        squareShape.addcomponent(square, color, 'gray')
        screen.register_shape(color, squareShape)

def neighborhood(grid, r, c):
    neighbors = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    rows = len(grid)
    cols = len(grid[0])
    sum = 0
    for n in neighbors:
        rr = (r + n[0]) % rows
        cc = (c + n[1]) % cols
        if grid[rr][cc] != grid[r][c]:
            sum = sum + 1
    return sum

def ising(rows, cols, passes):
    tortoise = turtle.Turtle()
    screen = tortoise.getscreen()
    screen.setup(cols * SCALE + 20, rows * SCALE + 20)
    screen.setworldcoordinates(0, 0, cols, rows)
    tortoise.speed(0)
    tortoise.hideturtle()
    tortoise.up()
    turtle.tracer(5)
    createSquares(screen, ['blue', 'lightblue'])

    grid = []
    for r in range(rows):
        row = []
        for c in range(cols):
            if random.random() < PROP_UP:
                row.append(1)
                drawSquare((r, c), 'blue', tortoise)
            else:
                row.append(-1)
                drawSquare((r, c), 'lightblue', tortoise)
        grid.append(row)

    # Simulate 'passes' iterations of the Metropolis algorithm.
    for p in range(passes):
        for r in range(rows):
            for c in range(cols):
                oldSpin = grid[r][c]
```

```
                newSpin = -oldSpin
                spinSum = neighborhood(grid, r, c)
                oldEnergy = spinSum
                newEnergy = 4 - spinSum
                energyDiff = newEnergy - oldEnergy
                if energyDiff <= 0 or random.random() < math.exp(-energyDiff / T):
                    grid[r][c] = newSpin
                    if newSpin == 1:
                        drawSquare((r, c), 'blue', tortoise)
                    else:
                        drawSquare((r, c), 'lightblue', tortoise)

    screen.update()
    screen.exitonclick()

def main():
    ising(30, 30, 1000)

main()
```

Project 9.3

```
import turtle
import random
import math

def adjacent(grid, x, y):
    maxX = len(grid[0])
    maxY = len(grid)
    offsets = [(-1, -1), (-1, 0), (-1, 1), (1, -1), (1, 0), (1, 1), (0, -1), (0, 1)]
    for offset in offsets:
        xx = x + offset[0]
        yy = y + offset[1]
        if xx in range(maxX) and yy in range(maxY) and grid[xx][yy] == 1:
            return True
    return False

def distance(p, q):
    return math.sqrt((p[0] - q[0]) ** 2 + (p[1] - q[1]) ** 2)

def dla(particles, maxX, maxY):
    tortoise = turtle.Turtle()
    screen = tortoise.getscreen()
    screen.setworldcoordinates(0, 0, maxX - 1, maxY - 1)
    tortoise.pencolor("blue")
    tortoise.speed(0)
    screen.tracer(10)
    tortoise.hideturtle()
    tortoise.up()
    oX = maxX // 2
    oY = maxY // 2
    tortoise.goto(oX, oY)
    tortoise.dot()
```

```
    grid = []
    for x in range(maxX):
        grid.append([0] * maxY)
    grid[oX][oY] = 1

    radius = 0
    for p in range(particles):
        angle = random.random() * 2 * math.pi
        x = oX + int((radius + 1) * math.cos(angle))
        y = oY + int((radius + 1) * math.sin(angle))

        for moves in range(200):
            r = random.random()
            if r < 0.25:
                x = x - 1
            elif r < 0.5:
                y = y - 1
            elif r < 0.75:
                x = x + 1
            else:
                y = y + 1

            if adjacent(grid, x, y):
                tortoise.goto(x, y)
                tortoise.dot()
                grid[x][y] = 1
                dist = distance((oX, oY), (x, y))
                if dist > radius:
                    radius = dist
                break

    screen.exitonclick()

def main():
    dla(10000, 200, 200)

main()
```

CHAPTER 10

Self-similarity and recursion

10.1 EXERCISE SOLUTIONS

Section 10.1

10.1.1
```
def tree(tortoise, length, depth):
    if depth <= 1:
        tortoise.forward(length)
        tortoise.backward(length)
    else:
        angle1 = random.randrange(10, 61)
        angle2 = random.randrange(10, 61)
        shrink1 = random.random() * 0.25 + 0.5
        shrink2 = random.random() * 0.25 + 0.5
        tortoise.forward(length)
        tortoise.left(angle1)
        tree(tortoise, length * shrink1, depth - 1)
        tortoise.right(angle1 + angle2)
        tree(tortoise, length * shrink2, depth - 1)
        tortoise.left(angle2)
        tortoise.backward(length)
```

10.1.2
```
def quadkoch(tortoise, length, depth):
    if depth <= 0:
        tortoise.forward(length)
    else:
        koch(tortoise, length / 3, depth - 1)
        tortoise.left(90)
        koch(tortoise, length / 3, depth - 1)
        tortoise.right(90)
        koch(tortoise, length / 3, depth - 1)
        tortoise.right(90)
        koch(tortoise, length / 3, depth - 1)
        tortoise.left(90)
        koch(tortoise, length / 3, depth - 1)
```

10.1.4
```
def kochSnowFlake(tortoise, length, depth, sides):
    for side in range(sides):
        koch(tortoise, length, depth)
        tortoise.right(360 / sides)
```

10.1.5
```
def drawTriangle(tortoise, p1, p2, p3):
    tortoise.up()
```

```
        tortoise.goto(p1)
        tortoise.down()
        tortoise.goto(p2)
        tortoise.goto(p3)
        tortoise.goto(p1)

    def midPoint(p1, p2):
        return ((p1[0] + p2[0]) / 2.0, (p1[1] + p2[1]) / 2.0)

    def sierpinski(tortoise, p1, p2, p3, depth):
        if depth <= 0:
            drawTriangle(tortoise, p1, p2, p3)
        else:
            sierpinski(tortoise, p1, midPoint(p1, p2), midPoint(p1, p3), depth - 1)
            sierpinski(tortoise, p2, midPoint(p2, p3), midPoint(p2, p1), depth - 1)
            sierpinski(tortoise, p3, midPoint(p3, p1), midPoint(p3, p2), depth - 1)
```

10.1.6

```
    def hilbert(tortoise, reverse, depth):
        if depth >= 0:
            if reverse:
                angle = -90
            else:
                angle = 90
            tortoise.right(angle)
            hilbert(tortoise, not reverse, depth - 1)
            tortoise.forward(10)
            tortoise.left(angle)
            hilbert(tortoise, reverse, depth - 1)
            tortoise.forward(10)
            hilbert(tortoise, reverse, depth - 1)
            tortoise.left(angle)
            tortoise.forward(10)
            hilbert(tortoise, not reverse, depth - 1)
            tortoise.right(angle)
```

10.1.7

```
    def drawRectangle(tortoise, upperLeft, width, color):
        tortoise.pencolor(color)
        tortoise.fillcolor(color)
        tortoise.up()
        tortoise.goto(upperLeft)
        tortoise.down()
        tortoise.begin_fill()
        for side in range(4):
            tortoise.forward(width)
            tortoise.right(90)
        tortoise.end_fill()

    def carpet(tortoise, upperLeft, width, depth):
        (x, y) = upperLeft
        drawRectangle(tortoise, (x + width / 3, y + width / 3), width / 3, 'blue')
        if depth > 0:
            carpet(tortoise, upperLeft, width / 3, depth - 1)
            carpet(tortoise, (x + width / 3, y), width / 3, depth - 1)
            carpet(tortoise, (x + 2 * width / 3, y), width / 3, depth - 1)
            carpet(tortoise, (x, y + width / 3), width / 3, depth - 1)
```

```
        carpet(tortoise, (x + 2 * width / 3, y + width / 3), width / 3, depth - 1)
        carpet(tortoise, (x, y + 2 * width / 3), width / 3, depth - 1)
        carpet(tortoise, (x + width / 3, y + 2 * width / 3), width / 3, depth - 1)
        carpet(tortoise, (x + 2 * width / 3, y + 2 * width / 3), width / 3, depth - 1)
```

Section 10.2

10.2.1
```
def sum(n):
    if n <= 1:
        return 1
    return sum(n - 1) + n
```

10.2.2
```
def factorial(n):
    if n <= 1:
        return 1
    else:
        return n * factorial(n - 1)
```

10.2.3
```
def power(a, n):
    if n == 0:
        return 1
    return a * power(a, n - 1)
```

10.2.4
```
def minList(data):
    if data == []:
        return None
    if len(data) == 1:
        return data[0]
    return minList(data[0], minList(data[1:]))
```

10.2.5
```
def length(data):
    if data == []:
        return 0
    return 1 + length(data[1:])
```

10.2.6
```
def gcd(m, n):
    if n == 0:
        return m
    r = m % n
    return gcd(n, r)
```

10.2.7
```
def reverse(text):
    if text == '':
        return ''
    return reverse(text[1:]) + text[0]
```

10.2.8
```
def digit2string(d):
    return chr(ord('0') + d)

def int2string(n):
    if n < 10:
        return digit2string(n)
    return int2string(n // 10) + digit2string(n % 10)
```

10.2.9
```
def countUpper(s):
    if s == '':
        return 0
    if s[0].isupper():
        return 1 + countUpper(s[1:])
    return countUpper(s[1:])
```

10.2.10
```
def equal(list1, list2):
    if len(list1) + len(list2) == 0:
        return True
    if len(list1) != len(list2) or list1[0] != list2[0]:
        return False
    return equal(list1[1:], list2[1:])
```

10.2.11
```
def subsets(n):
    if n == 0:
        return [[]]
    withoutN = subsets(n - 1)
    withN = []
    for subset in withoutN:
        withN.append([n] + subset)
    return withoutN + withN
```

10.2.12
```
def licensePlates(length, letters, numbers):
    if length == 0:
        return [' ']
    shorterPlates = licensePlates(length - 1, letters, numbers)
    plates = []
    for letter in letters:
        for number in numbers:
            for shorterPlate in shorterPlates:
                plates.append(letter + shorterPlate + number)
    return plates
```

Section 10.4

10.4.1
```
def linearSearch(data, target):
    if len(data) == 0:                    # base case 1: not found
        return False
    if target == data[0]:                 # base case 2: found
        return True
    return linearSearch(data[1:], target)  # recursive case
```

10.4.2 No, because the slicing operation, which creates a new list, requires linear time by itself. So now the function is performing a linear number of linear-time operations, resulting in a quadratic-time algorithm.

10.4.3
```
def linearSearch(data, target, last):
    if (last < 0) or (last >= len(data)):  # base case 1: not found
        return -1
    if target == data[last]:               # base case 2: found
        return last
    return linearSearch(data, target, last - 1) # recursive case
```

10.4.5
```
def sumSearch(data, total, first):
    if (first < 0) or (first >= len(data)):
        return -1
    if data[first] >= total:
        return first
    return sumSearch(data, total - data[first], first + 1)
```

Section 10.5

10.5.1
```
def power(a, n):
```

```
    if n == 0:
        return 1
    elif n == 1:
        return a
    elif n % 2 == 0:
        return power(a, n // 2) ** 2
    else:
        return a * power(a, n // 2) ** 2
```

10.5.2
```
def fibonacci(n):
    if n <= 2:
        return 1
    return fibonacci(n - 1) + fibonacci(n - 2)
```

10.5.3 No, this would not work. When the length of `prices` is 2, this method of dividing the list would place the entire list in the left half and an empty list in the right half. Therefore, a list of length 2 is never actually divided and we never reach the base case.

10.5.4
```
def profit(prices, first, last):
    if last <= first:
        return 0

    midIndex = (first + last) // 2
    leftProfit = profit(prices, first, midIndex)
    rightProfit = profit(prices, midIndex + 1, last)

    buy = min(prices[first:midIndex + 1])
    sell = max(prices[midIndex + 1: last + 1])
    midProfit = sell - buy
    return max(leftProfit, rightProfit, midProfit)
```

10.5.5
```
def profit(prices, first, last):
    if last <= first:
        return (first, first)
    midIndex = (first + last) // 2
    (leftBuy, leftSell) = profit(prices, first, midIndex)
    (rightBuy, rightSell) = profit(prices, midIndex + 1, last)

    buy = midIndex
    for index in range(first, midIndex):
        if prices[index] < prices[buy]:
            buy = index

    sell = midIndex + 1
    for index in range(midIndex + 2, last + 1):
        if prices[index] > prices[sell]:
            sell = index

    if prices[leftSell] - prices[leftBuy] >= max(prices[rightSell] - prices[rightBuy],
                                                 prices[sell] - prices[buy]):
        return (leftBuy, leftSell)
    if prices[rightSell] - prices[rightBuy] >= prices[sell] - prices[buy]:
        return (rightBuy, rightSell)
    return (buy, sell)
```

10.5.6
```
def linearSearch(data, target, first, last):
    if first > last:                    # base case 1: not found
        return -1
```

```
        midIndex = (first + last) // 2
        if target == data[midIndex]:          # base case 2: found
            return midIndex
        index1 = linearSearch(data, target, first, midIndex - 1)
        if index1 >= 0:
            return index1
        return linearSearch(data, target, midIndex + 1, last)
```

10.5.7
```
def dfs(grid, source, dest, path):
    (row, col) = source
    rows = len(grid)
    columns = len(grid[0])

    if (row < 0) or (row >= rows) \textbackslash
      or (col < 0) or (col >= columns) \textbackslash
      or (grid[row][col]== BLOCKED) \textbackslash
      or (grid[row][col] == VISITED):    # dead end (base case)
        return False                     #   so return False

    if source == dest:                   # dest found (base case)
        return True                      #   so return True

    grid[row][col] = VISITED             # visit this cell
    path.append(source)

    if dfs(grid, (row, col + 1), dest, path):
        return True
    if dfs(grid, (row + 1, col), dest, path):
        return True
    if dfs(grid, (row, col - 1), dest, path):
        return True
    if dfs(grid, (row - 1, col), dest, path):
        return True

    path.pop()
    return False
```

10.5.8
```
def numPaths(n, row, column):
    if (row > n - 1) or (column > n - 1):
        return 0
    if (row, column) == (n - 1, n - 1):
        return 1
    return numPaths(n, row + 1, column) + numPaths(n, row, column + 1)
```

10.5.9
```
def binary(prefix, length):
    if len(prefix) == length:
        return [prefix]

    return binary(prefix + '0', length) + binary(prefix + '1', length)
```

Section 10.6

10.6.1
```
def drawLSystem(tortoise, string, angle, distance):
    for symbol in string:
        if symbol == 'F':
            george.forward(distance)
```

```
            elif symbol == '+':
                george.right(angle)
            elif symbol == '-':
                george.left(angle)
```

10.6.2
```
drawLSystem('F-F++F-F++F-F++F-F++F-F++F-F', 60, 20)

drawLSystem('FX-YF-FX+YF-FX-YF+FX+YF-FX-YF-FX+YF+FX-YF+FX+YF', 90, 20)
```

10.6.3
```
def lsystem(axiom, productions, depth, position, heading,
            angle, distance):
    """Produce a string from an L-system and draw it.

     Parameters:
        axiom: a string representing an L-system axiom
        productions: a dictionary containing L-system productions
        depth: the number of times the productions should be applied
        position: the initial position for the turtle
        heading: the initial heading for the turtle
        angle: the angle to turn on a '+' or '-' symbol
        distance: the distance to move on an 'F' symbol

    Return value: None
    """

    newString = applyProductions(axiom, productions, depth)
    drawLSystem(newString, angle, distance, position, heading)
```

10.6.4
```
lsystem('F', {'F': 'F-F++F-F'}, 4, (-400, 0), 0, 60, 10)

lsystem('FX', {'X': 'X-YF', 'Y':'FX+Y'}, 12, (0, 0), 0, 90, 5)

lsystem('F-F-F-F', {'F': 'F-F+F+FF-F-F+F'}, 3, (-100, -100), 0, 90, 3)

lsystem('F-F-F-F', {'F': 'FF-F-F-F-F-F+F'}, 3, (0, -200), 0, 90, 5)
```

10.6.5 The axiom should be changed to `F++F++F`.

10.2 PROJECT SOLUTIONS

Project 10.1

```
import turtle

def derive(string, productions, depth):
    if depth <= 0:
        return string

    newString = ''
    for symbol in string:
        if symbol in productions:
            newString = newString + productions[symbol]
        else:
            newString = newString + symbol
    print(newString)
    return derive(newString, productions, depth - 1)
```

```
# def drawLSystem(tortoise, string, angle, distance):
#     stack = []
#     for symbol in string:
#         if symbol == 'F':
#             tortoise.forward(distance)
#         elif symbol == '+':
#             tortoise.right(angle)
#         elif symbol == '-':
#             tortoise.left(angle)
#         elif symbol == '[':
#             pos = tortoise.position()
#             head = tortoise.heading()
#             stack.append((pos, head))
#         elif symbol == ']':
#             pos, head = stack.pop()
#             tortoise.up()
#             tortoise.goto(pos)
#             tortoise.down()
#             tortoise.setheading(head)

def drawLSystem(tortoise, string, startIndex, angle, distance):
    index = startIndex
    while index < len(string):
        symbol = string[index]
        if symbol == 'F':
            tortoise.forward(distance)
        elif symbol == '+':
            tortoise.right(angle)
        elif symbol == '-':
            tortoise.left(angle)
        elif symbol == '[':
            pos = tortoise.position()
            head = tortoise.heading()

            index = drawLSystem(tortoise, string, index + 1, angle, distance)

            tortoise.up()
            tortoise.goto(pos)
            tortoise.down()
            tortoise.setheading(head)
        elif symbol == ']':
            return index
        index = index + 1

    return len(string)

def lsystem(axiom, productions, depth, angle, distance, position, heading):
    george = turtle.Turtle()
    screen = george.getscreen()
    screen.tracer(100)
    george.speed(0)
    george.hideturtle()

    george.up()
```

```
    george.goto(position)
    george.down()
    george.setheading(heading)

    newString = derive(axiom, productions, depth)
    drawLSystem(george, newString, 0, angle, distance)

    screen.update()
    screen.exitonclick()

def main():
    lsystem('X', {'X': 'F[-X]+X', 'F': 'FF'}, 4, 30, 20, (0, -200), 90)
    lsystem('X', {'X': 'F-[[X]+X]+F[+FX]-X', 'F': 'FF'}, 5, 25, 6, (0, -300), 90)
    lsystem('F', {'F': 'FF-[-F+F+F]+[+F-F-F]'}, 4, 22.5, 10, (0, -300), 90)
    lsystem('X', {'X': 'F[+X]F[-X]+X', 'F': 'FF'}, 6, 30, 5, (0, -300), 90)
    lsystem('H', {'H': 'HFX[+H][-H]', 'X': 'X[-FFF][+FFF]FX'}, 6, 25.7, 5, (0, -300), 90)

main()
```

Project 10.2

```
import image
import math

import sys
sys.setrecursionlimit(10000)

def distance(p, q):
    sum = 0
    for index in range(len(p)):
        sum = sum + (p[index] - q[index]) ** 2
    return math.sqrt(sum)

def getCentroid(points):
    x = 0
    y = 0
    for point in points:
        x = x + point[0]
        y = y + point[1]
    return (x / len(points), y / len(points))

def meanDistance(points, centroid):
    total = 0
    for point in points:
        total = total + distance(point, centroid)
    return total / len(points)

def meanSquareDistance(points, centroid):
    total = 0
    for point in points:
        total = total + distance(point, centroid) ** 2
    return total / len(points)

def stdDevDistance(points, centroid):
```

```
    return math.sqrt(meanSquareDistance(points, centroid)
           - meanDistance(points, centroid) ** 2)

def measureDistrict(map, x, y, color, points):
    if (x < 0) or (x >= map.width()) or (y < 0) or (y >= map.height()):
        return (1, 0)
    current = map.get(x, y)
    if current == (255, 255, 255):  # visited
        return (0, 0)
    if distance(current, color) > 80:
        return (1, 0)

    map.set(x, y, (255, 255, 255))    # mark as visited (white)
    points.append((x, y))

    (p1, a1) = measureDistrict(map, x + 1, y, color, points)
    (p2, a2) = measureDistrict(map, x - 1, y, color, points)
    (p3, a3) = measureDistrict(map, x, y + 1, color, points)
    (p4, a4) = measureDistrict(map, x, y - 1, color, points)

    return (p1 + p2 + p3 + p4, a1 + a2 + a3 + a4 + 1)

def compactness(imageName, districts):
    state = image.Image(file = imageName, title = 'State')
    state.show()

    totalMeanDistance = 0
    totalStdDevDistance = 0
    totalCompactness = 0
    for (x, y) in districts:
        color = state.get(x, y)
        points = []
        perimeter, area = measureDistrict(state, x, y, color, points)
        if len(points) > 0:
            meanDist = meanDistance(points, getCentroid(points))
            stdDevDist = stdDevDistance(points, getCentroid(points))
            compactness = 4 * math.pi * area / perimeter ** 2
            totalMeanDistance = totalMeanDistance + meanDist
            totalStdDevDistance = totalStdDevDistance + stdDevDist
            totalCompactness = totalCompactness + compactness
        else:
            districts.remove((x, y))
            print(x, y)
        state.update()

    image.mainloop()

    return totalMeanDistance / len(districts),
           totalStdDevDistance / len(districts),
           totalCompactness / len(districts)

def main():
    ne1 = [(96, 48), (160, 48), (180, 56)]
    ne2 = [(96, 48), (168, 40), (176, 80)]
```

```
    nm1 = [(96, 24), (96, 80), (120, 144)]
    nm2 = [(48, 24), (48, 112), (128, 64)]
    sc1 = [(48, 40), (48, 8), (88, 32), (80, 80), (112, 104), (160, 48), (132, 104)]
    sc2 = [(32, 16), (72, 16), (56, 56), (112, 40), (96, 96), (160, 64), (144, 104)]
    oh1 = [(75, 120), (30, 60), (80, 47), (125, 40), (150, 30), (75, 75), (100, 75),
           (120, 70), (150, 60), (10, 120), (75, 100), (140, 100), (30, 130), (20, 150),
           (50, 170), (80, 140)]
    oh2 = [(20, 50), (60, 70), (100, 70), (120, 50), (130, 60), (130, 40), (150, 40),
           (150, 85), (20, 100), (50, 120), (75, 110), (120, 120), (75, 150), (40, 160),
           (10, 140), (10, 160)]
    ar1 = [(60, 10), (50, 120), (85, 70), (140, 40)]
    ar2 = [(30, 80), (70, 40), (140, 50), (90, 130)]

    meanDistance, meanStdDev, meanCompactness = compactness('ne1.gif', ne1)

    print('Mean distance to centoids:', meanDistance)
    print('Mean standard deviation of distance to centroids:', meanStdDev)
    print('Mean compactness:', meanCompactness)

main()
```

Project 10.3

```
import turtle
import random
import matplotlib.pyplot as pyplot

BLOCKED = 0  # site is blocked
OPEN = 1     # site is open and empty
FULL = 2     # site is open and full

SCALE = 16   # drawing scale

def drawSquare(pos, color, tortoise):
    (row, column) = pos
    screen = tortoise.getscreen()
    rows = int(screen.canvheight / screen.yscale)
    row = rows - row - 1
    tortoise.shape(color)
    tortoise.up()
    tortoise.goto(column, row + 1)
    tortoise.stamp()

def drawGrid(grid, tortoise):
    rows = len(grid)
    columns = len(grid[0])
    for row in range(rows):
        for col in range(columns):
            if grid[row][col] == BLOCKED:
                drawSquare((row, col), 'black', tortoise)
            else:
                drawSquare((row, col), 'white', tortoise)

def createSquares(screen, colors):
```

```
    square = ((0, 0), (0, SCALE), (SCALE, SCALE), (SCALE, 0))
    for color in colors:
        squareShape = turtle.Shape('compound')
        squareShape.addcomponent(square, color, 'gray')
        screen.register_shape(color, squareShape)

def randomGrid(rows, columns, p):
    grid = []
    for r in range(rows):
        row = []
        for c in range(columns):
            if random.random() < p:
                row.append(OPEN)
            else:
                row.append(BLOCKED)
        grid.append(row)
    return grid

def dfs(grid, row, col, tortoise, draw):
    rows = len(grid)
    columns = len(grid[0])
    if row < 0 or row >= rows or col < 0 or col >= columns:
        return
    if grid[row][col] == BLOCKED or grid[row][col] == FULL:
        return
    grid[row][col] = FULL
    if draw:
        drawSquare((row, col), 'blue', tortoise)
    dfs(grid, row + 1, col, tortoise, draw)   # south
    dfs(grid, row, col - 1, tortoise, draw)   # west
    dfs(grid, row, col + 1, tortoise, draw)   # east
    dfs(grid, row - 1, col, tortoise, draw)   # north

def percolates(grid, draw):
    rows = len(grid)
    columns = len(grid[0])
    if draw:
        tortoise = turtle.Turtle()
        screen = tortoise.getscreen()
        screen.setup(columns * SCALE + 20, rows * SCALE + 20)
        screen.setworldcoordinates(0, 0, columns, rows)
        screen.tracer(5)
        tortoise.hideturtle()
        createSquares(screen, ['black', 'white', 'blue'])
        drawGrid(grid, tortoise)
    else:
        tortoise = None

    for col in range(columns):
        dfs(grid, 0, col, tortoise, draw)

    if draw:
        screen.update()
        screen.exitonclick()
```

```
    if FULL in grid[rows - 1]:
        return True
    return False

def percMonteCarlo(rows, columns, p, trials):
    count = 0
    for t in range(trials):
        grid = randomGrid(rows, columns, p)
        if percolates(grid, False):
            count = count + 1

    return count / trials

def percPlot(rows, columns, minP, maxP, stepP, trials):
    p = minP
    pList = []
    percList = []
    while p < maxP:
        print(p)
        pList.append(p)
        percList.append(percMonteCarlo(rows, columns, p, trials))
        p = p + stepP

    pyplot.plot(pList, percList)
    pyplot.show()

def main():
    rows = 20
    columns = 20

    percPlot(rows, columns, 0, 1, 0.01, 10000)

main()
```

CHAPTER 11

Organizing data

11.1 EXERCISE SOLUTIONS

Section 11.1

11.1.2
```
def spellcheck(fileName):
    dictFile = open('/usr/share/dict/words', 'r', encoding = 'utf-8')
    words = []
    for word in dictFile:
        words.append(word[:-1])
    dictFile.close()
    words.sort()

    punctuation = ''
    for code in range(256):
        if not chr(code).isalpha():
            punctuation = punctuation + chr(code)

    count = 0
    inputFile = open(fileName, 'r', encoding = 'utf-8')
    for line in inputFile:
        wordsInLine = line.split()
        for word in wordsInLine:
            word = word.strip(punctuation)
            index = binarySearch(words, word, 0, len(words) - 1)
            if index == -1:
                count = count + 1
    inputFile.close()
    return count
```

11.1.3
```
import matplotlib.pyplot as pyplot, time

def searchPlot(minLength, maxLength, step):
    data = list(range(maxLength))
    lsTimes = []
    bsTimes = []
    for size in range(minLength, maxLength + 1, step):
        slice = data[:size]

        start = time.time()
        linearSearch(slice, -1)
```

```
        finish = time.time()
        lsTimes.append(finish - start)

        start = time.time()
        binarySearch(slice, -1)
        finish = time.time()
        bsTimes.append(finish - start)

    pyplot.plot(range(minLength, maxLength + 1, step), lsTimes, label = 'Linear search')
    pyplot.plot(range(minLength, maxLength + 1, step), bsTimes, label = 'Binary search')
    pyplot.legend(loc = 'upper center')
    pyplot.xlabel('len(keys)')
    pyplot.ylabel('Time in seconds')
    pyplot.show()
```

11.1.4 Since the function guesses in the middle of its range from `left` to `right` each time, the worst case is for the secret number to be `n`. When `n` is 1, the function obviously returns 1 guess. When `n` is 2, the function requires 2 guesses in the worst case. Now consider the case when `n` is 4 and `secret` is 4. The function will initially guess `(1 + 4) // 2` = `2`. Since this is too low, it will set `left` to 3 and guess `(3 + 4) // 2` = `3`. This is also too low, so it sets `left` to 4 and guesses `(4 + 4) // 2` = `4`, which is correct. So it took 3 guesses. Looked at another way, after its first incorrect guess, the function is left with a range of 2 numbers (3 and 4). We already know that it takes 2 guesses in the worst case to guess one of two numbers, so its must take one more guess, or 3 guesses, to guess one of 4 numbers. This rationale will continue for increasing powers of 2. So, for `n` $= 2^i$, $i + 1 = \log_2 n + 1$ guesses are necessary in the worst case.

Section 11.2

11.2.1 No, every list requires the same number of comparisons.

11.2.2 The minimum number of swaps is 0, if the list is already sorted. The maximum number is $n - 1$. An example of a list that produces the maximum number of swaps is the "almost sorted" list `[2, 3, 4, 5, 6, 7, 8, 9, 10, 1]`.

11.2.3 In terms of elementary steps, `selectionSort2` is less efficient because it requires that the list be traversed twice in each iteration of the loop. However, because `min` and `index` are highly optimized, it may actually be faster.

11.2.4
```
def insert(data, item):
    index = 0
    while (index < len(data)) and (data[index] < item):
        index = index + 1
    data.insert(index, item)
```

11.2.5
```
import urllib.request as web

def readQuakes():
    url = 'http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_month.csv'
    quakeFile = web.urlopen(url)
    header = quakeFile.readline()

    ids = []
    data = []
    for line in quakeFile:
        line = line.decode('utf-8')
        row = line.split(",")
```

```
            latitude = float(row[1])
            longitude = float(row[2])
            depth = float(row[3])
            magnitude = float(row[4])
            data.append((latitude, longitude, depth, magnitude))
            ids.append(row[11])
        quakeFile.close()
        return ids, data
```

11.2.6
```
def queryQuakes(ids, data):
    key = input('Earthquake ID (q to quit): ')
    while key != 'q':
        if key == 'list':
            print('    ID            Location         Magnitude  Depth')
            print('----------  --------------------  ---------  -----')
            for index in range(len(ids)):
                print('{0:<10}  {1:^22}  {2:>9}  {3:>5}'.format(ids[index],
                                data[index][:2], data[index][2], data[index][3]))
            print()
        else:
            index = binarySearch(ids, key, 0, len(ids) - 1)
            if index >= 0:
                print('Location: ' + str(data[index][:2]) + '\n' +
                      'Magnitude: ' + str(data[index][3]) + '\n' +
                      'Depth: ' + str(data[index][2]) + '\n')
            else:
                print('An earthquake with that ID was not found.')
        key = input('Earthquake ID (q to quit): ')
```

11.2.7
```
import urllib.request as web

def readQuakes():
    url = 'http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_month.csv'
    quakeFile = web.urlopen(url)
    header = quakeFile.readline()

    quakes = []
    for line in quakeFile:
        line = line.decode('utf-8')
        row = line.split(",")
        latitude = float(row[1])
        longitude = float(row[2])
        depth = float(row[3])
        magnitude = float(row[4])
        id = row[11]
        quakes.append((id, latitude, longitude, depth, magnitude))
    quakeFile.close()
    return quakes

def swap(data, i, j):
    temp = data[i]
    data[i] = data[j]
    data[j] = temp

def selectionSort(data):
```

```
    n = len(data)
    for start in range(n - 1):
        minIndex = start
        for index in range(start + 1, n):
            if data[index][0] < data[minIndex][0]:
                minIndex = index
        swap(data, start, minIndex)

def binarySearch(data, target, left, right):
    if left > right:              # base case 1: not found
        return -1
    mid = (left + right) // 2
    key = data[mid][0]
    if target == key:             # base case 2: found
        return mid
    if target < key:              # recursive cases
        return binarySearch(data, target, left, mid - 1)  # 1st half
    return binarySearch(data, target, mid + 1, right)     # 2nd half

def queryQuakes(quakes):
    key = input('Earthquake ID (q to quit): ')
    while key != 'q':
        if key == 'list':
            print('    ID             Location         Magnitude  Depth')
            print('----------  --------------------  ---------  -----')
            for index in range(len(quakes)):
                print('{0:<10}  {1:^22}  {2:>9}  {3:>5}'.format(quakes[index][0],
                      quakes[index][1:3], quakes[index][3], quakes[index][4]))
            print()
        else:
            index = binarySearch(quakes, key, 0, len(quakes) - 1)
            if index >= 0:
                print('Location: ' + str(quakes[index][1:3]) + '\n' +
                      'Magnitude: ' + str(quakes[index][4]) + '\n' +
                      'Depth: ' + str(quakes[index][3]) + '\n')
            else:
                print('An earthquake with that ID was not found.')
        key = input('Earthquake ID (q to quit): ')
```

11.2.8

```
def sieve(n):
    prime = [False, False] + [True] * (n - 1)
    for index in range(2, n // 2):
        if prime[index]:
            for multiple in range(2 * index, n + 1, index):
                prime[multiple] = False
    primes = []
    for index in range(2, n + 1):
        if prime[index]:
            primes.append(index)
    return primes
```

Section 11.3

11.3.1 A list in reverse order requires the worst case number of comparisons in an insertion sort. For example, `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`. This list requires $n(n+1)/2-1 = (10)(11)/2-1 = 54$ comparisons.

11.3.2 A list that is already sorted requires the best case number of comparisons in an insertion sort. For example, `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`. This list requires only $n-1 = 9$ comparisons.

11.3.3
```
def timing():
    dictFile = open('/usr/share/dict/words', 'r')
    words = []
    for word in dictFile:
        words.append(word[:-1])
    dictFile.close()

    words1 = words[:]
    start = time.time()
    words1.sort()
    finish = time.time()
    print('The sort method required', finish - start, 'seconds.')

    start = time.time()
    insertionSort(words2)
    finish = time.time()
    print('The insertion sort required', finish - start, 'seconds.')
```
The `sort` method will take a fraction of a second while insertion sort will take about ten minutes.

11.3.4
```
import matplotlib.pyplot as pyplot, random, time

def sortPlot(minLength, maxLength, step):
    data = list(range(maxLength))
    random.shuffle(data)
    ssTimes = []
    isTimes = []
    tsTimes = []
    for size in range(minLength, maxLength + 1, step):
        slice = data[:size]
        begin = time.time()
        selectionSort(slice)
        end = time.time()
        ssTimes.append(end - begin)

        slice = data[:size]
        begin = time.time()
        insertionSort(slice)
        end = time.time()
        isTimes.append(end - begin)

        slice = data[:size]
        begin = time.time()
        slice.sort()
        end = time.time()
        tsTimes.append(end - begin)
```

```
pyplot.plot(range(minLength, maxLength + 1, step), ssTimes, label = 'Selection sort')
pyplot.plot(range(minLength, maxLength + 1, step), isTimes, label = 'Insertion sort')
pyplot.plot(range(minLength, maxLength + 1, step), tsTimes, label = 'Built-in sort')
pyplot.legend(loc = 'upper center')
pyplot.xlabel('len(data)')
pyplot.ylabel('Time in seconds')
pyplot.show()
```

11.3.5 Selection sort is not a stable sort. To see why not, consider the tiny list `[2, 2, 1]`. Selection sort will swap the first `2` with the `1`, resulting in the two values of `2` being in the opposite order as they were originally.

On the other hand, insertion sort is stable since it never swaps values.

11.3.6
```
def bubbleSort(data):
    for lastIndex in range(len(data) - 1, 0, -1):
        for index in range(lastIndex):
            if data[index] > data[index + 1]:
                swap(data, index, index + 1)
```

11.3.7
```
def bubbleSort(data):
    lastIndex = len(data) - 1
    swapped = True
    while swapped:
        swapped = False
        for index in range(lastIndex):
            if data[index] > data[index + 1]:
                swap(data, index, index + 1)
                swapped = True
        lastIndex = lastIndex - 1
```

11.3.8
```
def insertionSort(keys, data):
    n = len(data)
    for insertIndex in range(1, n):
        itemToInsert = keys[insertIndex]
        itemToInsert2 = data[insertIndex]
        index = insertIndex - 1
        while index >= 0 and keys[index] > itemToInsert:
            keys[index + 1] = keys[index]
            data[index + 1] = data[index]
            index = index - 1
        keys[index + 1] = itemToInsert
        data[index + 1] = itemToInsert2
```

Section 11.4

11.4.1 When $n = 100$, selection sort is about $100/log_2 100 \approx 15$ times slower. When $n = 1000$, selection sort is about $1000/log_2 1000 \approx 100$ times slower. When $n = 1$ million, selection sort is about $1000000/log_2 1000000 \approx 50,172$ times slower.

11.4.4 If line 18 were comparing two items with the same value, the item in the right list would be appended to the merged list ahead of the item from the left list.

11.4.5 (a) With linear search, the time complexity is n.

(b) To use binary search, the list must be sorted first. So the time complexity is $n \log_2 n + \log_2 n$, which is $n \log_2 n$ asymptotically.

(c) n linear searches have time complexity n^2.

(d) n binary searches have time complexity $n \log_2 n + n \log_2 n$, which is $n \log_2 n$ asymptotically.

(e) Sorting the list and then using binary search is only worthwhile if you plan to perform sufficiently many searches. If the number is proportional to n, using binary search is definitely better.

11.4.6 (a) The linear search option is better because $n^2/2 + k \cdot \log_2 n = 524,288 + 100 \cdot 10 = 525,288$ is greater than $k \cdot n = 100 \cdot 1024 = 102,400$.

(b) The linear search option is still better because $n^2/2 + k \cdot \log_2 n = 524,288 + 500 \cdot 10 = 529,288$ is greater than $k \cdot n = 500 \cdot 1024 = 512,000$.

(c) The binary search option is better because $n^2/2 + k \cdot \log_2 n = 524,288 + 1000 \cdot 10 = 534,288$ is less than $k \cdot n = 1000 \cdot 1024 = 1,024,000$.

11.4.7
```
def merge(leftFileName, rightFileName, mergedFileName):
    """Merge two sorted files into one sorted file.

    Parameters:
        leftFileName: name of a sorted file
        rightFileName: name of another sorted file
        mergedFileName: name of the merged file

    Return value: None
    """

    leftFile = open(leftFileName, 'r', encoding = 'utf-8')
    rightFile = open(rightFileName, 'r', encoding = 'utf-8')
    mergedFile = open(mergedFileName, 'w', encoding = 'utf-8')

    leftItem = leftFile.readline().rstrip()
    rightItem = rightFile.readline().rstrip()
    while leftItem != '' and rightItem != '':
        print('leftItem =', leftItem, 'rightItem =', rightItem)
        if leftItem <= rightItem:
            mergedFile.write(leftItem + '\n')    # left value is smaller
            leftItem = leftFile.readline().rstrip()
        else:
            mergedFile.write(rightItem + '\n')   # right value is smaller
            rightItem = rightFile.readline().rstrip()

    if leftItem == '':             # items remaining in right
        mergedFile.write(rightItem + '\n')
        for rightItem in rightFile:
            mergedFile.write(rightItem + '\n')
    else:                                  # items remaining in left
        mergedFile.write(leftItem + '\n')
        for leftItem in leftFile:
            mergedFile.write(leftItem + '\n')

    leftFile.close()
    rightFile.close()
    mergedFile.close()
```

11.2 PROJECT SOLUTIONS

Project 11.2

```
# Extended BST implementation

KEY = 0
DATA = 1
LEFT = 2
RIGHT = 3

def insert(root, key, data):
    current = root
    while current != []:
        if key <= current[KEY]:
            current = current[LEFT]
        else:
            current = current[RIGHT]
    current.extend([key, data, [], []])

def search(root, key):
    current = root
    while current != [] and current[KEY] != key:
        if key < current[KEY]:
            current = current[LEFT]
        else:
            current = current[RIGHT]
    if current != []:
        return current[DATA]
    else:
        return None

# Recursive functions

def insert(root, key, data):
    if root == []:
        root.extend([key, data, [], []])
    elif key <= root[KEY]:
        insert(root[LEFT], key, data)
    else:
        insert(root[RIGHT], key, data)

def search(root, key):
    if root == []:
        return None
    if root[KEY] == key:
        return root[DATA]
    if key < root[KEY]:
        return search(root[LEFT], key)
    else:
        return search(root[RIGHT], key)

def bstSort(root):
    if root == []:
        return []
```

```
return bstSort(root[LEFT]) + [(root[KEY], root[DATA])] + bstSort(root[RIGHT])
```

CHAPTER 12

Networks

12.1 EXERCISE SOLUTIONS

Section 12.1

12.1.8
```
def readGraph(fileName):
    graphFile = open(fileName, 'r', encoding = 'utf-8')
    graph = { }
    for line in graphFile:
        edge = line.split()
        if edge[0] in graph:
            graph[edge[0]].append(edge[1])
        else:
            graph[edge[0]] = [edge[1]]
        if edge[1] in graph:
            graph[edge[1]].append(edge[0])
        else:
            graph[edge[1]] = [edge[0]]
    graphFile.close()
    return graph
```

12.1.9
```
def readGraphDirected(fileName):
    graphFile = open(fileName, 'r', encoding = 'utf-8')
    graph = { }
    for line in graphFile:
        edge = line.split()
        if edge[0] in graph:
            graph[edge[0]].append(edge[1])
        else:
            graph[edge[0]] = [edge[1]]
    graphFile.close()
    return graph
```

12.1.10
```
def maxDegree(graph):
    maximum = 0
    for node in graph:
        if len(graph[node]) > maximum:
            maximum = len(graph[node])
    return maximum
```

12.1.11
```
def avgDegree(graph):
    totalDegree = 0
    for node in graph:
```

```
            totalDegree = totalDegree + len(graph[node])
        return totalDegree / len(graph)
```

Section 12.2

12.2.1 Ted, Cathy, Christina, Beth, Dave, Kevin, Ryder, Tyler, Vanessa, Amelia, Nick, Caroline, Lillian

12.2.2 Caroline, Amelia, Lillian, Nick, Beth, Cathy, Dave, Ted, Christina, Kevin, Ryder, Tyler, Vanessa

12.2.3
```
def bfs(network, source):
    distance = { }
    predecessor = { }
    for node in network:
        distance[node] = float('inf')
        predecessor[node] = None
    distance[source] = 0
    queue = [source]
    while queue != []:
        front = queue.pop(0)
        for neighbor in network[front]:
            if distance[neighbor] == float('inf'):
                distance[neighbor] = distance[front] + 1
                predecessor[neighbor] = front
                queue.append(neighbor)
    return distance, predecessor
```

12.2.4
```
def distance(network, source, dest):
    distances = bfs(network, source)
    return distances[dest]
```

12.2.5
```
def connected(network):
    source = list(network.keys())[0]
    visited = { }
    for node in network:
        visited[node] = False
    visited[source] = True
    queue = [source]
    while queue != [ ]:
        front = queue.pop(0)
        for neighbor in network[front]:
            if not visited[neighbor]:
                visited[neighbor] = True
                queue.append(neighbor)
    for node in network:
        if not visited[node]:
            return False
    return True
```

12.2.6
```
def dfs(network, source, visited):
    if source in visited:
        return

    visited.append(source)

    for neighbor in network[source]:
```

```
            dfs(network, neighbor, visited)

    def connected(network):
        visited = []
        source = list(network.keys())[0]
        dfs(network, source, visited)

        for node in network:
            if node not in visited:
                return False
        return True
```

Section 12.3

12.3.1
```
def averageClusteringCoefficient(network):
    totalCC = 0
    for node in network:
        totalCC = totalCC + clusteringCoefficient(network, node)
    return totalCC / len(network)
```

12.3.2
```
def averageDistance(network):
    totalDistance = 0
    n = len(network)
    nodes = list(network.keys())
    for index1 in range(len(nodes) - 1):
        source = nodes[index1]
        dist, pred = bfs(network, source)
        for index2 in range(index1 + 1, len(nodes)):
            dest = nodes[index2]
            if dist[dest] == float('inf'):
                totalDistance = totalDistance + n
            else:
                totalDistance = totalDistance + dist[dest]
    return totalDistance / (n * (n - 1) / 2)
```

12.3.3
```
def closenessCentrality(network, node):
    totalDistance = 0
    dist, pred = bfs(network, node)
    for other in network:
        if dist[other] == float('inf'):
            dist[other] = len(network)
        totalDistance = totalDistance + dist[other]
    return totalDistance
```

12.3.4
```
def minBC(network):
    n = len(network)
    minimum = n * n
    minNode = 0
    for node in network:
        bc = closenessCentrality(network, node)
        if bc < minimum:
            minimum = bc
            minNode = node
    return minNode
```

12.3.5
```
def degreeDistribution(graph):
    counts = {}
```

```
for node in graph:
    degree = len(graph[node])
    if degree in counts:
        counts[degree] = counts[degree] + 1
    else:
        counts[degree] = 1

for degree in counts:
    counts[degree] = counts[degree] / len(graph)

degrees = list(counts.keys())
fractions = list(counts.values())

pyplot.plot(degrees, fractions)
pyplot.xlabel('Degree')
pyplot.ylabel('Fraction of nodes')
pyplot.xlim(0, max(degrees) + 10)
pyplot.show()
```

Section 12.4

12.4.1
```
graph = randomGraph(30, 50/((50 * 49) / 2))
```

12.4.2
```
def avgCCRandom(n, p, trials):
    totalCC = 0
    for trial in range(trials):
        graph = randomGraph(n, p)
        cc = averageClusteringCoefficient(graph)
        totalCC = totalCC + cc

    return totalCC / trials
```

12.4.3
```
def avgDistanceRandom(n, p, trials):
    totalDistance = 0
    for trial in range(trials):
        graph = randomGraph(n, p)
        d = averageDistance(graph)
        totalDistance = totalDistance + d

    return totalDistance / trials
```

12.4.4
```
def degreeDistributionRandom(n, p, trials):
    counts = {}
    for trial in range(trials):
        graph = randomGraph(n, p)
        for node in graph:
            degree = len(graph[node])
            if degree in counts:
                counts[degree] = counts[degree] + 1
            else:
                counts[degree] = 1

    for degree in counts:
        counts[degree] = counts[degree] / (n * trials)

    degrees = list(counts.keys())
```

```
        fractions = list(counts.values())

        pyplot.plot(degrees, fractions)
        pyplot.xlabel('Degree')
        pyplot.ylabel('Fraction of nodes')
        pyplot.show()
```

12.4.5
```
    def dfs(network, source, visited):
        if source in visited:
            return

        visited.append(source)

        for neighbor in network[source]:
            dfs(network, neighbor, visited)

    def connected(network):
        visited = []
        source = list(network.keys())[0]
        dfs(network, source, visited)

        for node in network:
            if node not in visited:
                return False
        return True

    def connectedRandom(n, minp, maxp, stepp, trials):
        fracConnected = []
        pValues = []
        p = minp
        while p <= maxp:
            count = 0
            for trial in range(trials):
                graph = randomGraph(n, p)
                if connected(graph):
                    count = count + 1
            fracConnected.append(count / trials)
            pValues.append(p)
            p = p + stepp

        pyplot.plot(pValues, fracConnected)
        pyplot.xlabel('$p$')
        pyplot.ylabel('Fraction connected')
        pyplot.show()

    # There should be a phase transition around p = n/log n.
```

12.2 PROJECT SOLUTIONS

Project 12.1

```
import random

def readGraph(fileName):
    graphFile = open(fileName, 'r', encoding = 'utf-8')
```

```
    graph = { }
    for line in graphFile:
        edge = line.split()
        if edge[0] in graph:
            graph[edge[0]].append(edge[1])
        else:
            graph[edge[0]] = [edge[1]]
        if edge[1] in graph:
            graph[edge[1]].append(edge[0])
        else:
            graph[edge[1]] = [edge[0]]
    graphFile.close()
    return graph

def diffusion(network, source, p):
    totalAdopted = 0
    adopted = { }
    for node in network:
        adopted[node] = False
    adopted[source] = True
    queue = [source]
    while queue != [ ]:
        front = queue.pop(0)
        for neighbor in network[front]:
            if not adopted[neighbor]:
                if random.random() < p:
                    adopted[neighbor] = True
                    totalAdopted = totalAdopted + 1
                    queue.append(neighbor)
    return totalAdopted

def rateSeed(network, source, p, trials):
    total = 0
    for trial in range(trials):
        adopted = diffusion(network, source, p)
        total = total + adopted
    return total / trials

def maxInfluence(network, p, trials):
    ratings = { }
    for node in network:
        rating = rateSeed(network, node, p, trials)
        if rating in ratings:
            ratings[rating].append(node)
        else:
            ratings[rating] = [node]

    allRatings = list(ratings.keys())
    return ratings[max(allRatings)]

def main():
    graph = readGraph('facebook_small1.txt')
    for i in range(10):
        winner = rateAllNodes(graph, 0.05, 500)
```

```
        print(winner)

main()
```

Project 12.2

```
#!/usr/bin/env python3.3

import matplotlib.pyplot as pyplot
import random

def readGraph(fileName):
    graphFile = open(fileName, 'r', encoding = 'utf-8')
    graph = { }
    for line in graphFile:
        edge = line.split()
        if edge[0] in graph:
            graph[edge[0]].append(edge[1])
        else:
            graph[edge[0]] = [edge[1]]
        if edge[1] in graph:
            graph[edge[1]].append(edge[0])
        else:
            graph[edge[1]] = [edge[0]]
    graphFile.close()
    return graph

def infection(network, p, vaccinated = None):
    source = random.choice(list(network.keys()))
    totalInfected = 0
    infected = { }
    immune = { }
    for node in network:
        infected[node] = False
        immune[node] = False
    infected[source] = True
    if vaccinated != None:
        for node in vaccinated:
            immune[node] = True
    queue = [source]
    while queue != [ ]:
        front = queue.pop(0)
        for neighbor in network[front]:
            if not infected[neighbor] and not immune[neighbor]:
                if random.random() < p:
                    infected[neighbor] = True
                    totalInfected = totalInfected + 1
                    queue.append(neighbor)
    return totalInfected

def getHubs(network):
    degrees = { }
    for node in network:
        degree = len(network[node])   # clusteringCoefficient(network, node)
```

```
            if degree in degrees:
                degrees[degree].append(node)
            else:
                degrees[degree] = [node]

    allDegrees = list(degrees.keys())
    allDegrees.sort()
    nodes = []
    for degree in allDegrees:
        nodes.extend(degrees[degree])

    return nodes

def vaccinationSim(network, randomSim, p, trials, vaccinations):
    if randomSim:
        vaccinationOrder = list(network.keys())
        random.shuffle(vaccinationOrder)
    else:
        vaccinationOrder = getHubs(network)

    index = -1
    vaccinated = []
    infecteds = []
    for vaccIndex in range(vaccinations):
        totalInfected = 0
        for trial in range(trials):
            infected = infection(network, p, vaccinated)
            totalInfected = totalInfected + infected
        vaccinated.append(vaccinationOrder[index])
        index = index - 1
        infecteds.append(totalInfected / trials)
    return infecteds

def main():
    graph = readGraph('facebook_small1.txt')
    vaccinations = 80
    trials = 500
    p = 0.1
    randInfecteds = vaccinationSim(graph, True, p, trials, vaccinations)
    hubInfecteds = vaccinationSim(graph, False, p, trials, vaccinations)

    pyplot.plot(range(vaccinations), randInfecteds, label = 'Random vaccinations')
    pyplot.plot(range(vaccinations), hubInfecteds, label = 'Vaccinating hubs')
    pyplot.legend()
    pyplot.xlabel('Number of vaccinations')
    pyplot.ylabel('Number infected')
    pyplot.show()

main()
```

Project 12.3

```
import matplotlib.pyplot as pyplot
```

```
def createGraph(filename):
    """Note: adjacency list entries are labeled with movie."""

    castFile = open(filename, 'r', encoding = 'utf-8')
    graph = {}   # with movies
    for line in castFile:
        line = line.strip()
        actors = line.split('\t')
        movie = actors[0]
        actors = actors[1:]
        for actor in actors:
            if actor not in graph:
                graph[actor] = []
            for costar in actors:
                if costar != actor and costar not in graph[actor]: # DON'T DUPLICATE ACTORS
                    graph[actor].append((costar, movie))
    castFile.close()
    return graph

def bfs(network, source):
    """Note: predecessors are (actor, movie)."""

    visited = { }
    distance = { }
    predecessor = { }
    for node in network:
        visited[node] = False
        distance[node] = float('inf')
        predecessor[node] = None
    visited[source] = True
    distance[source] = 0
    queue = [source]
    while queue != [ ]:
        front = queue.pop(0)
        for neighbor in network[front]:
            actor, movie = neighbor
            if not visited[actor]:
                visited[actor] = True
                distance[actor] = distance[front] + 1
                predecessor[actor] = (front, movie)
                queue.append(actor)
    return distance, predecessor

def printPath(graph, start, end, predecessor):
    print()
    current = end
    count = 1
    while current != start:
        pred = predecessor[current]
        name, movie = pred
        print(str(count) + '.', current , 'was in' , '"' + movie + '"' , 'with' , name)
        current = name
        count = count + 1
```

```
def oracle(graph):
    print()
    start = input('Start actor (q to quit): ')
    while start != 'q':
        end = input('End actor: ')
        print()
        distances, predecessors = bfs(graph, start)
        distance = distances[end]
        print(start, 'and', end, 'have distance', distance)
        printPath(graph, start, end, predecessors)
        print()
        start = input('Start actor (q to quit): ')

def degreeDistribution(graph):
    counts = {}
    for actor in graph:
        degree = len(graph[actor])
        if degree in counts:
            counts[degree] = counts[degree] + 1
        else:
            counts[degree] = 1

    n = len(graph)
    for degree in counts:
        counts[degree] = counts[degree] / n

    degrees = list(counts.keys())
    fractions = list(counts.values())

    pyplot.plot(degrees, fractions)
    pyplot.xlabel('Degree')
    pyplot.ylabel('Fraction of nodes')
    pyplot.xlim(0, max(degrees) + 10)
    pyplot.show()

    degrees = list(counts.keys())
    degrees.sort()
    bigDegree = degrees[-10]
    for actor in graph:
        degree = len(graph[actor])
        if degree >= bigDegree:
            print(actor, 'has degree', degree)

def baconNumbers(graph, actor):
    distances, predecessors = bfs(graph, actor)

    upper = 8
    counts = [0] * (upper + 2)
    for name in distances:
        distance = distances[name]
        if distance <= upper:
            counts[distance] = counts[distance] + 1
        else:
            counts[upper + 1] = counts[upper + 1] + 1
```

```
    print('Bacon Number   Frequency')
    print('------------   ---------')
    for number in range(len(counts) - 1):
        print('{:^12}   {:>9}'.format(number, counts[number]))
    print('  infinity     {:>9}'.format(counts[-1]))
    print()

def main():
    graph = createGraph('movies2013.txt')
    baconNumbers(graph, 'Kevin Bacon')
    degreeDistribution(graph)
    oracle(graph)

main()
```

CHAPTER 13

Abstract data types

13.1 EXERCISE SOLUTIONS

Section 13.1

13.1.1 In the `Turtle` class, the `xcor`, `ycor`, and `heading` methods are accessors because they return the turtle's current position and heading without modifying the turtle. Most of the methods that we commonly use in turtle graphics are mutator methods. For example, the `forward/backward`, `left/right`, and `goto` methods (to name just a few) are mutators because they each change some aspect of the turtle.

13.1.2 In the `list` class, the `index` and `count` methods are accessors while the `append`, `insert`, and `pop` methods are mutators.

13.1.3
```
def round(self):
    self._a = round(self._a)
    self._b = round(self._b)
```

13.1.4
```
def main():
    tally = Pair()
    votes = input('Enter votes (q to quit): ')
    while votes != 'q':
        votes = votes.split()
        votes = Pair(int(votes[0]), votes[1])
        tally = tally.add(votes)
        votes = input('Enter votes (q to quit): ')
    print(tally.get())
```

13.1.5
```
def addSplit(times, split):
    if len(times) > 0:
        elapsed = times[-1].getSecond() + split
    else:
        elapsed = split
    times.append(Pair(split, elapsed))

def main():
    times = []
    addSplit(times, 23)
    addSplit(times, 43)
    addSplit(times, 24)
    for time in times:
        print(time.get())
```

13.1.6
```
import matplotlib.pyplot as pyplot
```

```
    def plotAltitudes(altData):
        times = []
        altitudes = []
        for pair in altData:
            times.append(pair.getFirst())
            altitudes.append(pair.getSecond())
        pyplot.plot(range(len(times)), altitudes)
        pyplot.xticks(range(len(times)), times)
        pyplot.xlabel('Time')
        pyplot.ylabel('Altitude')
        pyplot.show()

    def main():
        altData = [Pair('01:00', 5000), Pair('01:01', 5200), Pair('01:02', 5800)]
        plotAltitudes(altData)

    main()
```

13.1.7
```
    def distance(point1, point2):
        return math.sqrt((point1.getFirst() - point2.getFirst()) ** 2
                         + (point1.getSecond() - point2.getSecond()) ** 2)
```

13.1.8
```
    def averageDistance(points, base):
        """Compute the average distance between a point and a list of points.

        Parameter:
            points: a list of Pair objects
            base: a Pair object

        Return value:
            the average distance between base and points
        """

        n = len(points)
        if n == 0:
            return None
        sum = 0
        for point in points:
            sum = sum + distance(base, point)
        return sum / n
```

13.1.9
```
    class Pair:
        def draw(self, tortoise, color):
            tortoise.up()
            tortoise.goto(self._x, self._y)
            tortoise.down()
            tortoise.pencolor(color)
            tortoise.dot(8)

    def closestPairs(points):
        closestDistance = distance(points[0], points[1])
        closestPair = (points[0], points[1])
        for index1 in range(1, len(points) - 1):
            for index2 in range(index1 + 1, len(points)):
                distance = distance(points[index1], points[index2])
```

```
                if distance < closestDistance:
                    closestPair = (points[index1], points[index2])
                    closestDistance = distance
        return closestPair

    def farthestPairs(points):
        farthestDistance = distance(points[0], points[1])
        farthestPair = (points[0], points[1])
        for index1 in range(1, len(points) - 1):
            for index2 in range(index1 + 1, len(points)):
                distance = distance(points[index1], points[index2])
                if distance > farthestDistance:
                    farthestPair = (points[index1], points[index2])
                    farthestDistance = distance
        return farthestPair
```

13.1.10

```
class Pair:
    def __init__(self, a = 0, b = 0):
        self._pair = [a, b]

    def getFirst(self):
        return self._pair[0]

    def getSecond(self):
        return self._pair[1]

    def get(self):
        return tuple(self._pair)

    def add(self, pair2):
        sumA = self._pair[0] + pair2._pair[0]
        sumB = self._pair[1] + pair2._pair[1]
        return Pair(sumA, sumB)

    def subtract(self, pair2):
        diffA = self._pair[0] - pair2._pair[0]
        diffB = self._pair[1] - pair2._pair[1]
        return Pair(diffA, diffB)

    def set(self, a, b):

        self._pair = [a, b]

    def scale(self, scalar):
        self.set(self._pair[0] * scalar, self._pair[1] * scalar)
```

13.1.11

```
class BankAccount:
    def __init__(self, balance):
        self._balance = balance

    def getBalance(self):
        return self._balance

    def deposit(self, amount):
        self._balance = self._balance + amount
```

```
        def withdraw(self, amount):
            self._balance = self._balance - amount
```

13.1.12
```
def main():
    amount = float(input('Initial balance? '))
    account = BankAccount(amount)
    response = input('(D)eposit, (W)ithdraw, or (Q)uit? ')
    while response != 'q':
        if response in 'dD':
            amount = float(input('Amount = '))
            account.deposit(amount)
            print('Your balance is now ${0:<4.2f}'.format(account.getBalance()))
        elif response in 'wW':
            amount = float(input('Amount = '))
            account.withdraw(amount)
            print('Your balance is now ${0:<4.2f}'.format(account.getBalance()))
        response = input('(D)eposit, (W)ithdraw, or (Q)uit? ')
```

13.1.13
```
class President:
    def __init__(self, name):
        self._name = name
        self._party = ''
        self._state = ''
        self._religion = ''
        self._age = 0

    def getName(self):
        return self._name

    def getParty(self):
        return self._party

    def getState(self):
        return self._state

    def getReligion(self):
        return self._religion

    def getAge(self):
        return self._age

    def setName(self, name):
        self._name = name

    def setParty(self, party):
        self._party = party

    def setState(self, state):
        self._state = state

    def setReligion(self, religion):
        self._religion = religion

    def setAge(self, age):
```

```
            self._age = age
```

13.1.14
```
def getPresidents():
    presFile = open('presidents.txt', 'r', encoding = 'utf-8')
    presidents = []
    for line in presFile:
        line = line.strip()
        values = line.split('\t')
        name = values[0]
        party = values[1]
        state = values[2]
        religion = values[3]
        age = int(values[4])
        president = President(name)
        president.setParty(party)
        president.setState(state)
        president.setReligion(religion)
        president.setAge(age)
        presidents.append(president)
    presFile.close()
    return presidents

def olderPresidents(presidents, age):
    print('Name                 Age')
    print('-------------------- ---')
    for president in presidents:
        if president.getAge() >= age:
            print('{0:<20} {1:>3}'.format(president.getName(), president.getAge()))
```

13.1.15
```
class Movie:
    def __init__(self, title, year, actors):
        self._title = title
        self._year = year
        self._actors = actors

    def getTitle(self):
        return self._title

    def getYear(self):
        return self._year

    def getActors(self):
        return self._actors

    def setTitle(self, newTitle):
        self._title = newTitle

    def setYear(self, newYear):
        self._year = newYear

    def setActors(self, newActors):
        self._actors = newActors

    def addActor(self, actor):
        self._actors.append(actor)
```

```
        def commonActors(self, otherMovie):
            for actor in self._actors:
                if actor in otherMovie.getActors():
                    return True
            return False
```

13.1.16
```
class Senator:
    def __init__(self, name, party, state):
        self._name = name
        self._party = party
        self._state = state
        self._committees = []

    def getName(self):
        return self._name

    def getParty(self):
        return self._party

    def getState(self):
        return self._state

    def getCommittees(self):
        return self._committees

    def addCommittee(self, committee):
        self._committees.append(committee)
```

13.1.17
```
def committeeMembership(senators, committee):
    members = []
    for senator in senators:
        if committee in senator.getCommittees():
            members.append(senator)
    return members

def main():
    senators = getSenators()
    for senator in senators:
        print(senator.getName() + '(' + senator.getParty() + ')')
        for committee in senator.getCommittees():
            print('    ' + committee)

    committee = input('Committee name (q to quit): ')
    while committee not in 'qQ':
        membership = committeeMembership(senators, committee)
        for senator in membership:
            print(senator.getName() + '(' + senator.getParty() + ')')
        committee = input('Committee name (q to quit): ')

main()
```

13.1.18
```
class Student:
    def __init__(self, name):
        self._name = name
        self._exams = []
```

```
        self._quizes = []
        self._labs = []
        self._papers = []

    def getName(self):
        return self._name

    def addExam(self, score):
        self._exams.append(score)

    def addQuiz(self, score):
        self._quizes.append(score)

    def addLab(self, score):
        self._labs.append(score)

    def addPaper(self, score):
        self._papers.append(score)

    def examAverage(self):
        if len(self._exams) > 0:
            return sum(self._exams) / len(self._exams)
        else:
            return 0

    def quizAverage(self):
        if len(self._quizes) > 0:
            return sum(self._quizes) / len(self._quizes)
        else:
            return 0

    def labAverage(self):
        if len(self._labs) > 0:
            return sum(self._labs) / len(self._labs)
        else:
            return 0

    def paperAverage(self):
        if len(self._papers) > 0:
            return sum(self._papers) / len(self._papers)
        else:
            return 0

    def grade(self):
        return self.examAverage() * 0.5 + self.quizAverage() * 0.1 \
              + self.labAverage() * 0.2 + self.paperAverage() * 0.2
```

13.1.19

```
class Dataset:
def __init__(self):
self._size = 0
self._min = 0
self._max = 0
self._sum = 0

def add(self, x):
```

```
self._size += 1
if x < self._min:
self._min = x
if x > self._max:
self._max = x
self._sum = self._sum + x

def min(self):
return self._min

def max(self):
return self._max

def average(self):
return self._sum / self._size

def size(self):
return self._size
```

13.1.20
```
class Sequence:
    def __init__(self, seq, type, accID):
        self._sequence = seq   # 'DNA' or 'RNA' or 'AA'
        self._type = type
        self._accID = accID

    def reverse(self):
        """return new Sequence"""

    def translate(self):
        """return new sequence if dna"""

    def gc(self):
        """return GC content if dna"""
```

Section 13.2

13.2.1
```
def __mul__(self, pair2):
    return Pair(self[0] * pair2[0], self[1] * pair2[1])

def __truediv__(self, pair2):
    return Pair(self[0] / pair2[0], self[1] / pair2[1])
```

13.2.2
```
def __ne__(self, pair2):
    return not (self == pair2)

def __le__(self, pair2):
    return (self == pair2) or (self < pair2)

def __gt__(self, pair2):
    return pair2 < self

def __ge__(self, pair2):
    return (self == pair2) or (self > pair2)
```

13.2.3
```
def linearRegression(points):
    n = len(points) # number of points
```

```
        sumx = 0   # sum of x coordinates
        sumy = 0   # sum of y coordinates
        sumxy = 0  # sum of products of x and y coordinates
        sumxx = 0  # sum of squares of x coordinates
        for index in range(n):
            sumx = sumx + points[index][0]
            sumy = sumy + points[index][1]
            sumxy = sumxy + points[index][0] * points[index][1]
            sumxx = sumxx + points[index][0] * points[index][0]
        sumx2 = sumx ** 2   # square of sum of x coordinates

        m = (n * sumxy - sumx * sumy) / (n * sumxx - sumx2)  # slope
        b = (sumy - m * sumx) / n                            # y intercept
        return m, b
```

13.2.4
```
    def __str__(self):
        return self._name + ' ' + self._party

def statePresidents(presidents, state):
    for president in presidents:
        if president.getState() == state:
            print(president)
```

13.2.5
```
    def __lt__(self, otherPres):
        return self._age < otherPres._age
```

13.2.6
```
    def __str__(self):
        return self._name + ' (' + self._party + ')'
```

13.2.7
```
    def distance(point1, point2):
        return math.sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)
```

13.2.8
```
    def committeeMembership(senators, committee):
        members = []
        for senator in senators:
            if committee in senator.getCommittees():
                members.append(senator)
        return members

def main():
    senators = getSenators()
    for senator in senators:
        print(senator)
        for committee in senator.getCommittees():
            print('    ' + committee)

    committee = input('Committee name (q to quit): ')
    while committee not in 'qQ':
        membership = committeeMembership(senators, committee)
        for senator in membership:
            print(senator)
        committee = input('Committee name (q to quit): ')

main()
```

13.2.9
```
    def gcd(j, k):
        if j > k:
            j, k = k, j
```

```
    r = 1
    while r > 0:
        r = k % j
        k = j
        j = r

    return k

class Rational:
    def __init__(self, num = 0, den = 1):
        if num != 0:
            g = gcd(abs(num), abs(den))
        else:
            g = 1

        self.num = num // g
        self.den = den // g

        if self.den < 0:
            self.den = -self.den
            self.num = -self.num

    def __add__(self, other):
        num = self.num * other.den + other.num * self.den
        den = self.den * other.den
        return Rational(num, den)

    def __sub__(self, other):
        num = self.num * other.den - other.num * self.den
        den = self.den * other.den
        return Rational(num, den)

    def __mul__(self, other):
        num = self.num * other.num
        den = self.den * other.den
        return Rational(num, den)

    def __truediv__(self, other):
        num = self.num * other.den
        den = self.den * other.num
        return Rational(num, den)

    def __lt__(self, other):
        return self.num * other.den < other.num * self.den

    def __eq__(self, other):
        return self.num * other.den == other.num * self.den

    def __str__(self):
        return str(self.num) + '/' + str(self.den)

    def __le__(self, other):
        return self < other or self == other
```

Section 13.3

13.3.1
```
import bankaccount

def main():
    amount = float(input('Initial balance? '))
    account = bankaccount.BankAccount(amount)
    response = input('(D)eposit, (W)ithdraw, or (Q)uit? ')
    while response != 'q':
        if response in 'dD':
            amount = float(input('Amount = '))
            account.deposit(amount)
            print('Your balance is now ${0:<4.2f}'.format(account.getBalance()))
        elif response in 'wW':
            amount = float(input('Amount = '))
            account.withdraw(amount)
            print('Your balance is now ${0:<4.2f}'.format(account.getBalance()))
        response = input('(D)eposit, (W)ithdraw, or (Q)uit? ')
```

13.3.3
```
import president

def getPresidents():
    presFile = open('presidents.txt', 'r', encoding = 'utf-8')
    presidents = []
    for line in presFile:
        line = line.strip()
        values = line.split('\t')
        name = values[0]
        party = values[1]
        state = values[2]
        religion = values[3]
        age = int(values[4])
        president = president.President(name)
        president.setParty(party)
        president.setState(state)
        president.setReligion(religion)
        president.setAge(age)
        presidents.append(president)
    presFile.close()
    return presidents
```

Section 13.4

13.4.2
```
if newVelocity == self._velocity:
    if random.random() < 0.2:
        newVelocity = self._velocity + Vector((random.uniform(-0.1, 0.1),
                                               random.uniform(-0.1, 0.1)))
```

13.4.3
```
if random.random() < 0.5:
    self._velocity.turn(TURN_ANGLE)
else:
    self._velocity.turn(-TURN_ANGLE)
```

Section 13.5

13.5.1
```
    def __str__(self):
```

```
            if len(self) == 0:
                stackString = '| (empty) | '
            else:
                stackString = '| '
                for item in self._stack:
                    stackString = stackString + repr(item) + ' | '
            return stackString + '<- top'
```

13.5.2

```
import stack

def convert(number, base):
    if number < 0:
        numberString = '-'
        number = -number
    else:
        numberString = ''

    digitStack = stack.Stack()
    while number > 0:
        digit = number % base
        digitStack.push(digit)
        number = number // base

    while not digitStack.isEmpty():
        digit = digitStack.pop()
        if digit < 10:
            numberString = numberString + str(digit)
        else:
            numberString = numberString + chr(ord('A') + digit - 10)
    return numberString
```

13.5.4

```
def balanced(expression):
    parenStack = stack.Stack()
    for character in expression:
        if character == '(':
            parenStack.push(character)
        elif character == ')':
            if parenStack.isEmpty():
                print('Too many right parentheses.')
                return False
            else:
                parenStack.pop()
    if not parenStack.isEmpty():
        print('Too many left parentheses.')
        return False
    return True
```

13.5.5

```
import stack

BLOCKED = 0  # site is blocked
OPEN = 1     # site is open and not visited
VISITED = 2  # site is open and already visited

def canSearch(grid, row, col):
    rows = len(grid)
    columns = len(grid[0])
```

```
        if (row < 0) or (row >= rows) \
          or (col < 0) or (col >= columns) \
          or (grid[row][col]== BLOCKED) \
          or (grid[row][col] == VISITED):    # dead end (base case)
            return False                     #   so return False
        return True

    def dfs(grid, source, dest):
        dfsStack = stack.Stack()
        dfsStack.push(source)
        while not dfsStack.isEmpty():
            (row, col) = dfsStack.pop()
            if (row, col) == dest:
                return True
            if canSearch(grid, row, col):
                grid[row][col] = VISITED         # visit this cell
                dfsStack.push((row, col + 1))    # push east cell
                dfsStack.push((row + 1, col))    # push south cell
                dfsStack.push((row, col - 1))    # push west cell
                dfsStack.push((row - 1, col))    # push north cell
        return False
```

13.5.7

```
    import turtle
    from point import *

    class PairSet:
        def __init__(self):
            self._points = []

        def insert(self, x, y):
            self._points.append(Pair(x, y))

        def __len__(self):
            return len(self._points)

        def centroid(self):
            n = len(self)
            if n == 0:
                return None
            sum = Pair(0, 0)
            for point in self._points:
                sum = sum + point
            return sum / n

        def closestPairs(self):
            if len(self) < 2:
                return None, None
            closestDistance = self._points[0].distance(self._points[1])
            closestPair = (0, 1)
            for index1 in range(1, len(self) - 1):
                for index2 in range(index1 + 1, len(self)):
                    distance = self._points[index1].distance(self._points[index2])
                    if distance < closestDistance:
                        closestPair = (index1, index2)
```

```
                    closestDistance = distance
        return self._points[closestPair[0]], self._points[closestPair[1]]

    def farthestPairs(self):
        if len(self) < 2:
            return None, None
        farthestDistance = self._points[0].distance(self._points[1])
        farthestPair = (0, 1)
        for index1 in range(1, len(self) - 1):
            for index2 in range(index1 + 1, len(self)):
                distance = self._points[index1].distance(self._points[index2])
                if distance > farthestDistance:
                    farthestPair = (index1, index2)
                    farthestDistance = distance
        return self._points[farthestPair[0]], self._points[farthestPair[1]]

    def diameter(self):
        if len(self) < 2:
            return 0
        point1, point2 = self.farthestPairs()
        return point1.distance(point2)

    def draw(self, tortoise, color):
        for point in self._points:
            point.draw(tortoise, 'black')

def main():
    points = PairSet()
    inputFile = open('africa.txt', 'r', encoding = 'utf-8')
    for line in inputFile:
        values = line.split()
        longitude = float(values[0])
        latitude = float(values[1])
        points.insert(longitude, latitude)

    cpoint1, cpoint2 = points.closestPairs()
    fpoint1, fpoint2 = points.farthestPairs()
    center = points.centroid()

    george = turtle.Turtle()
    screen = george.getscreen()
    screen.setworldcoordinates(-37, -23, 37, 58)
    george.hideturtle()
    george.speed(0)
    screen.tracer(10)
    points.draw(george, 'black')
    cpoint1.draw(george, 'blue')
    cpoint2.draw(george, 'blue')
    fpoint1.draw(george, 'red')
    fpoint2.draw(george, 'red')
    center.draw(george, 'yellow')
    screen.update()
    screen.exitonclick()
```

```
if __name__ == '__main__':
    main()
```

Section 13.6

13.6.1
```
def _printTable(self):
    for index in range(self._size):
        print(str(index) + ': ' + str(self._table[index]))
```

13.6.2
```
def __str__(self):
    allItems = []
    for slot in self._table:
        if slot != None:
            allItems.append(slot)

    dictString = '{'
    for pair in allItems[:-1]:
        dictString = dictString + repr(pair[self._KEY]) + ': '
                                + repr(pair[self._VALUE]) + ', '
    if len(allItems) > 0:
        dictString = dictString + repr(allItems[-1][self._KEY]) + ': '
                                + repr(allItems[-1][self._VALUE])
    return dictString + '}'
```

13.6.3
```
def __contains__(self, key):
    index = self._hash(key)
    if self._table[index] != None and self._table[index][self._KEY] == key:
        return True
    return False
```

13.6.8
```
def _hash(self, key):
    sum = 0
    for character in key:
        sum = sum + ord(character)
    return sum % self._size
```

13.6.10
```
class Presidents:
    def __init__(self, numPresidents):
        self._list = [None] * numPresidents

    def __getitem__(self, number):
        if number >= 1 and number <= len(self._list):
            return self._list[number - 1]
        else:
            return None

    def __setitem__(self, number, president):
        if number >= 1 and number <= len(self._list):
            self._list[number - 1] = president

    def __str__(self):
        presList = ''
        for number in range(len(self._list)):
            if self._list[number] != None:
                presList = presList + str(number + 1) + '. '
                                    + str(self._list[number]) + '\n'
            else:
```

```
                presList = presList + str(number + 1) + '. ???\n'
        return presList

    def olderPresidents(self, age):
        print('Name                 Age')
        print('-------------------- ---')
        for president in self._list:
            if president.getAge() >= age:
               print('{0:<20} {1:>3}'.format(president.getName(), president.getAge()))
```

13.6.11

```
class BestPictures:
    def __init__(self, number = 87):
        self._pictures = [None] * number
        self._number = number

    def __getitem__(self, number):
        if number >= 1 and number <= self._number:
            return self._pictures[number - 1]
        return None

    def __setitem__(self, number, movie):
        if number >= 1 and number <= self._number:
            self._pictures[number - 1] = movie

    def __str__(self):
        picturesString = ''
        for number in range(1, self._number + 1):
            if self[number] != None:
                picturesString = picturesString + str(number) + '. '
                                                + self[number].getTitle() + '\n'
            else:
                picturesString = picturesString + str(number) + '. ???\n'
        return picturesString

    def commonActors(self, number1, number2):
        if (number1 < 1) or (number1 > self._number) or \
           (number2 < 1) or (number2 > self._number) or \
           (self[number1] == None) or (self[number2] == None):
            return None
        return self[number1].commonActors(self[number2])
```

13.6.12

```
class Roster:
    def __init__(self):
        self._students = []
        self._ids = []

    def _find(self, id):
        for index in range(len(self)):
            if self._ids[index] == id:
                return index
        return -1

    def __getitem__(self, id):
        index = self._find(id)
        if index >= 0:
```

```
                return self._students[index]
            return None

        def __setitem__(self, id, student):
            index = self._find(id)
            if index == -1:
                self._students.append(student)
                self._ids.append(id)
            else:
                self._students[index] = student

        def __len__(self):
            return len(self._students)

        def averageGrade(self):
            sum = 0
            for student in self._students:
                sum = sum + student.grade()
            return sum / len(self)

        def examAverage(self):
            sum = 0
            for student in self._students:
                sum = sum + student.examAverage()
            return sum / len(self)

        def __str__(self):
            rosterStr = ''
            for index in range(len(self)):
                rosterStr = rosterStr + '{0:<3}  {1:<15}  {2:>5.2f}\n' \
                                        .format(self._ids[index],
                                                self._students[index].getName(),
                                                self._students[index].grade())
            return rosterStr
```

13.2 PROJECT SOLUTIONS

Project 13.1

```
"""time.py"""

class Time:
    """A representation of a particular date and time."""

    def __init__(self, date, time):
        """Construct a Time object.

        Parameters:
            self: the Time object
            date: a string in YYYY?MM?DD format
            time: a string in HH?MM?SS.S...S format

        Return value: the new Time object
        """
```

```
        self._year = int(date[:4])
        self._month = int(date[5:7])
        self._day = int(date[8:10])
        self._hour = int(time[:2])
        self._minutes = int(time[3:5])
        self._seconds = float(time[6:])

    def duration(self, otherTime):
        """Return the number of seconds elapsed between self and otherTime."""

        seconds = otherTime._seconds - self._seconds
        seconds = seconds + (otherTime._minutes - self._minutes) * 60
        seconds = seconds + (otherTime._hour - self._hour) * 3600
        seconds = seconds + (otherTime._day - self._day) * 3600 * 24
        seconds = seconds + (otherTime._month - self._month) * 3600 * 24 * 30
        seconds = seconds + (otherTime._year - self._year) * 3600 * 24 * 30 * 365
        return seconds

    def date(self):
        """Return a YYYY-MM-DD string representation of the date."""

        return '{0:<4}-{1:0>2}-{2:0>2}'.format(self._year, self._month, self._day)

    def time(self):
        """Return a HH-MM-SS string representation of the time."""

        return '{0:0>2}:{1:0>2}:{2:0>4.1f}'.format(self._hour, self._minutes, self._seconds)

    def __str__(self):
        """Return a YYYY-MM-DD HH:MM:SS string representation of the date and time."""

        date = '{0:<4}-{1:0>2}-{2:0>2}'.format(self._year, self._month, self._day)
        time = '{0:0>2}:{1:0>2}:{2:0>4.1f}'.format(self._hour, self._minutes, self._seconds)
        return date + ' ' + time

"""point.py"""

import math

class Pair:
    """A two-dimensional point class."""

    def __init__(self, x = 0, y = 0, time = None):
        """Constructor initializes a Pair object to (x,y).

        Parameter:
            self: a Pair object

        Return value: None
        """

        self._x = x   # the point's x coordinate
        self._y = y   # the point's y coordinate
```

```
    self._time = time

def getX(self):
    """Return the x coordinate of self.

    Parameter:
        self: a Pair object

    Return value: the x coordinate of self
    """

    return self._x

def getY(self):
    """Return the y coordinate of self.

    Parameter:
        self: a Pair object

    Return value: the y coordinate of self
    """

    return self._y

def getXY(self):
    """Return a (x, y) tuple representing self.

    Parameter:
        self: a Pair object

    Return value: the (x, y) tuple representing self
    """

    return (self._x, self._y)

def time(self):
    return self._time

def set(self, x, y, time):
    """Set the x and y coordinates of self.

    Parameters:
        self: a Pair object
        x: a number representing a new x coordinate for self
        y: a number representing a new y coordinate for self

    Return value: None
    """

    self._x = x
    self._y = y
    self._time = time

def distance(self, otherPair):
```

```
        """Return the distance between self and otherPair.

        Parameters:
            self: a Pair object
            otherPair: another Pair object

        Return value: the distance between self and otherPair
        """

        diffX = self._x - otherPair.getX()
        diffY = self._y - otherPair.getY()
        return math.sqrt(diffX ** 2 + diffY ** 2)

    def __str__(self):
        """Return an '(x, y)' string representation of self.

        Parameter:
            self: a Pair object

        Return value: an '(x, y)' string representation of self
        """

        return '(' + str(self._x) + ', ' + str(self._y) + ')'

    def __add__(self, otherPair):
        """Return a new Pair object that is the
           sum of self and otherPair.

        Parameters:
            self: a Pair object
            otherPair: another Pair object

        Return value: a Pair that is the sum of self and otherPair
        """

        newX = self._x + otherPair.getX()
        newY = self._y + otherPair.getY()
        return Pair(newX, newY)

    def __sub__(self, otherPair):
        """Return a new Pair object that is the
           difference of self and otherPair.

        Parameters:
            self: a Pair object
            otherPair: another Pair object

        Return value: a Pair that is the difference of self and otherPair
        """

        newX = self._x - otherPair.getX()
        newY = self._y - otherPair.getY()
        return Pair(newX, newY)
```

```
"""track.py"""

from point import *
from time import *
import turtle
import math

class Track:
    """A sequence of geographical points with time stamps that track
       a moving object."""

    def __init__(self, name):
        self._name = name
        self._points = []

    def _distance(self, point1, point2):
        """Return the approximate distance (in km) between two geographical points."""

        kmPerLat = 111.0
        kmPerLong = 111.32 * math.cos(math.radians(abs(point1.getY() + point2.getY()) / 2))
        differenceLong = abs(point1.getX() - point2.getX())
        differenceLat = abs(point1.getY() - point2.getY())
        kmLong = differenceLong * kmPerLong
        kmLat = differenceLat * kmPerLat
        distanceKm = math.sqrt(kmLong ** 2 + kmLat ** 2)
        return distanceKm

    def _distanceIndices(self, index1, index2):
        """Return the approximate distance (in km) between the two points
           in the track with the given indices."""

        point1 = self._points[index1]
        point2 = self._points[index2]
        return _distance(point1, point2)

    def _speed(self, index1, index2):
        """Return the approximate speed traveled between the two points
           in the track with the given indices."""

        dist = self._distanceIndices(index1, index2)
        time1 = self._points[index1].time()
        time2 = self._points[index2].time()
        seconds = time1.duration(time2)
        speed = dist / (seconds / 3600)
        return speed

    def append(self, long, lat, date, time):
        """Add a point and time to the end of the track."""

        self._points.append(Pair(long, lat, Time(date, time)))

    def __len__(self):
        """Return the number of points on the track."""
```

```
        return len(self._points)

    def averageSpeed(self):
        """Return the average speed over the track."""

        if len(self) < 2:
            return 0
        sum = 0
        for index in range(1, len(self)):
            speed = self._speed(index - 1, index)
            sum = sum + speed
        return sum / (len(self) - 1)

    def totalDistance(self):
        """Return the total distance traversed on the track."""

        dist = 0
        for index in range(1, len(self)):
            dist = dist + self._distanceIndices(index - 1, index)
        return dist

    def diameter(self):
        """Return the distance between the two points that are
           farthest apart on the track."""

        if len(self) < 2:
            return 0
        farthestDistance = self._distanceIndices(0, 1)
        for index1 in range(1, len(self) - 1):
            for index2 in range(index1 + 1, len(self)):
                distance = self._distanceIndices(index1, index2)
                if distance > farthestDistance:
                    farthestDistance = distance
        return farthestDistance

    def closestDistance(self, location, error):
        """Find the closest distance a point on the track comes to the
           given point; return this distance and the time(s) when the
           track comes within error of this distance."""

        minDist = self._distance(location, self._points[0])
        distances = [(minDist, self._points[0].time())]
        for index in range(1, len(self)):
            distance = self._distance(location, self._points[index])
            distances.append((distance, self._points[index].time()))
            if distance < minDist:
                minDist = distance
        times = []
        for distance, time in distances:
            if abs(distance - minDist) <= error:
                times.append(time)
        return minDist, times

    def closestDistance2(self, location1, location2, error):
```

```
        minDist1 = self._distance(location1, self._points[0])
        minDist2 = self._distance(location2, self._points[0])
        distances = [(minDist1, self._points[0].time())]
        for index in range(1, len(self)):
            distance1 = self._distance(location1, self._points[index])
            distance2 = self._distance(location2, self._points[index])
            distances.append((distance1, self._points[index].time()))
            if distance1 < minDist1:
                minDist1 = distance1
            if distance2 < minDist2:
                minDist2 = distance2
        times = []
        for distance, time in distances:
            if abs(distance - minDist1) <= error:
                times.append(time)
        return minDist1 + minDist2, times

    def draw(self, degToPix):
        """Draw the track, using the degToPix function to convert each
           geographical point to an equivalent pixel location in the
           graphics window."""

        tortoise = turtle.Turtle()
        screen = tortoise.getscreen()
        tortoise.speed(0)
        tortoise.hideturtle()
        screen.tracer(10)
        tortoise.pencolor('blue')
        tortoise.up()
        tortoise.goto(degToPix(self._points[0]))
        tortoise.down()
        for p in self._points:
            tortoise.goto(degToPix(p))

"""muni.py"""

from point import *
from time import *
from track import *
import math
import turtle

MAP_FILENAME = 'muni.gif' # background image file name
MAP_WIDTH = 770           # width of background image muni.gif
MAP_HEIGHT = 657          # height of background image muni.gif
MIN_LONG = -122.5256      # longitude range represented in muni.gif
MAX_LONG = -122.3564
MIN_LAT = 37.7        # latitude range represented in muni.gif
MAX_LAT = 37.82

tracks = { }              # global dictionary of tracks
                          # (must be global so it is accessible in clickMap)

def degToPix(geoPair):
```

```
    """Convert a Pair object containing a geographical location to a tuple
       representing the equivalent pixel in the graphics window."""

    long = geoPair.getX()
    lat = geoPair.getY()
    x = -(MAP_WIDTH / 2) + MAP_WIDTH * ((long - MIN_LONG) / (MAX_LONG - MIN_LONG))
    y = -(MAP_HEIGHT / 2) + MAP_HEIGHT * ((lat - MIN_LAT) / (MAX_LAT - MIN_LAT))
    return (x, y)

def pixToDeg(turtlePair):
    """Convert a Pair object containing a pixel location in the graphics window
       to a tuple representing the equivalent geographical location."""

    x = turtlePair.getX()
    y = turtlePair.getY()
    long = MIN_LONG + ((x + MAP_WIDTH / 2) / MAP_WIDTH) * (MAX_LONG - MIN_LONG)
    lat = MIN_LAT + ((y + MAP_HEIGHT / 2) / MAP_HEIGHT) * (MAX_LAT - MIN_LAT)
    return (long, lat)

def readTracks(fileName):
    """Read tracking info from a file into a dictionary of Track objects."""

    tracksFile = open(fileName, 'r', encoding = 'utf-8')
    tracksDict = { }
    tracksFile.readline()
    for line in tracksFile:
        values = line.split(',')
        month = values[1][:2]
        day = values[1][3:5]
        year = values[1][6:10]
        date = year + '-' + month + '-' + day
        time = values[1][11:]
        name = values[2]
        long = float(values[3])
        lat = float(values[4])
        if name not in tracksDict:
            tracksDict[name] = Track(name)
        tracksDict[name].append(long, lat, date, time)
    tracksFile.close()
    return tracksDict

def clickMap(x, y):
    """Respond to a mouse click in the graphics window at position (x, y)
       by drawing the closest track and listing the times that the track
       comes within 100 meters of that position."""

    if (x < -MAP_WIDTH / 2) or (x > MAP_WIDTH / 2) or (y < -MAP_HEIGHT / 2) or \
       (y > MAP_HEIGHT / 2):
        return

    tortoise = turtle.Turtle()
    screen = tortoise.getscreen()
    screen.clear()                # clear the graphics window  (and mouse click binding)
    screen.bgpic(MAP_FILENAME)    # redraw the background map
```

```
    tortoise.pencolor('red')
    tortoise.up()
    tortoise.goto(x, y)
    tortoise.down()
    tortoise.dot(10)               # plot the clicked-upon point
    tortoise.dot(10)
    tortoise.hideturtle()
    screen.update()

    long, lat = pixToDeg(Pair(x, y))
    location = Pair(long, lat)   # convert (x, y) to (long, lat)

    # find closest track and the times when it comes
    #   within 100 meters of location

    trackNames = list(tracks.keys())
    minDist, minTimes = tracks[trackNames[0]].closestDistance(location, 0.1)
    for name in trackNames[1:]:
        distance, times = tracks[name].closestDistance(location, 0.1)  # 100 meters
        if distance < minDist:
            minDist = distance
            minName = name
            minTimes = times

    # write the times in the graphics window

    tortoise.up()
    tortoise.goto(MAP_WIDTH / 2 + 10, MAP_HEIGHT / 2)
    tortoise.pencolor('black')
    tortoise.write('Vehicle number ' + name, font = ('Helvetica', 12, 'bold'))
    tortoise.goto(MAP_WIDTH / 2 + 10, MAP_HEIGHT / 2 - 20)
    tortoise.write('Closest distance: {:4.2f} km'.format(minDist),
                   font = ('Helvetica', 12, 'bold'))
    tortoise.goto(MAP_WIDTH / 2 + 10, MAP_HEIGHT / 2 - 40)
    tortoise.write('Times:', font = ('Helvetica', 12, 'bold'))
    skip = 55
    for index in range(len(times)):
        if index == 0 or times[index - 1].duration(times[index]) >= 300:  # 5 minutes
            tortoise.goto(MAP_WIDTH / 2 + 10, MAP_HEIGHT / 2 - skip)
            tortoise.write('  ' + str(times[index].time()),
                           font = ('Helvetica', 10, 'normal'))
            skip = skip + 13

    tracks[minName].draw(degToPix)     # draw the track
    screen.onclick(clickMap)           # reassign this function to be called
                                       #   on a mouse click

def main():
    global tracks
    tracks = readTracks('muni_tracking.csv')   # read tracks into a dictionary of tracks

    george = turtle.Turtle()     # set up turtle window with background map
    screen = george.getscreen()
    george.hideturtle()
```

```
    screen.setup(1200, 800)
    screen.bgpic(MAP_FILENAME)   # draw the map in the center of the window
    screen.onclick(clickMap)     # set event handler to call clickMap when a click occurs
    screen.mainloop()            # enter main event loop to wait for mouse clicks

main()
```

Project 13.2

```
"""dictionary.py"""

KEY = 0
VALUE = 1

class Dictionary:
    def __init__(self):
        self._size = 11
        self._table = []
        for index in range(self._size):
            self._table.append([])

    def _hash(self, key):
        key = str(key)
        sum = 0
        for character in key:
            sum = sum + ord(character)
        return sum % self._size

    def _find(self, key):
        index = self._hash(key)
        slot = self._table[index]
        for slotIndex in range(len(slot)):
            if slot[slotIndex][KEY] == key:
                return slotIndex, slot
        return -1, slot

    def _printTable(self):
        tableString = ''
        for index in range(self._size):
            tableString = tableString + str(index) + ': ['
            for pair in self._table[index][:-1]:
                tableString = tableString + repr(pair[KEY]) + ': ' + repr(pair[VALUE]) + ', '
            if len(self._table[index]) > 0:
                pair = self._table[index][-1]
                tableString = tableString + repr(pair[KEY]) + ': ' + repr(pair[VALUE])
            tableString = tableString + ']\n'
        return tableString

    def __setitem__(self, key, value):
        slotIndex, slot = self._find(key)
        if slotIndex >= 0:
            slot[slotIndex] = (key, value)  # replace item
        else:
            slot.append((key, value))       # insert item
```

```
    def __getitem__(self, key):
        slotIndex, slot = self._find(key)
        if slotIndex >= 0:
            return slot[slotIndex][VALUE]
        else:
            raise KeyError('key was not found')

    def __delitem__(self, key):
        slotIndex, slot = self._find(key)
        if slotIndex >= 0:
            slot.pop(slotIndex)
        else:
            raise KeyError('key was not found')

    def __str__(self):
        allItems = self.items()

        tableString = '{'
        for pair in allItems[:-1]:
            tableString = tableString + repr(pair[KEY]) + ': ' + repr(pair[VALUE]) + ', '
        if len(allItems) > 0:
            tableString = tableString + repr(allItems[-1][KEY]) + ': '
                                      + repr(allItems[-1][VALUE])
        return tableString + '}'

    def __contains__(self, key):
        slotIndex, slot = self._find(key)
        return slotIndex != -1

    def items(self):
        allItems = []
        for slot in self._table:
            allItems.extend(slot)
        return allItems

    def keys(self):
        return [item[KEY] for item in self.items()]

    def values(self):
        return [item[VALUE] for item in self.items()]

"""mobility.py"""

from dictionary import *

def readFile(fileName):
    inputFile = open(fileName, 'r', encoding = 'utf-8')
    data = Dictionary()
    stateData = Dictionary()
    header = inputFile.readline()
    for line in inputFile:
        values = line.split('\t')
        czName = values[1]
```

```
        state = values[2]
        if values[6] != '':
            mobility = float(values[6])
        else:
            mobility = None
        data[czName + ', ' + state] = mobility
        if mobility != None:
            if state not in stateData:
                stateData[state] = []
            stateData[state].append((czName, mobility))

    inputFile.close()
    return data, stateData

def getSecond(pair):
    return pair[1]

def main():
    data, stateData = readFile('mobility_by_cz.txt')

    states = stateData.keys()
    states.sort()

    # Print all CZ's alphabetically by state then CZ name.

    for state in states:
        print(state)
        pairs = stateData[state]
        pairs.sort()
        for pair in pairs:
            print('  ' + pair[0] + ': {:>3.1f}%'.format(pair[1] * 100))

    # Print a table of state averages, organized alphabetically by state.

    stateAvg = Dictionary()
    print('State  Percent')
    print('-----  -------')
    for state in states:
        pairs = stateData[state]
        sum = 0
        for pair in pairs:
            sum = sum + pair[1]
        average = sum / len(pairs)
        stateAvg[state] = average
        print(' {0:<2}     {1:>4.1f}%'.format(state, average * 100))

    # Print five lowest and highest states.

    pairs = stateAvg.items()
    pairs.sort(key = getSecond)

    print()
    print('States with lowest upward mobility')
    print()
```

```
    print('State  Percent')
    print('-----  -------')
    for pair in pairs[:5]:
        print(' {0:<2}     {1:>4.1f}%'.format(pair[0], pair[1] * 100))

    print()
    print('States with higest upward mobility')
    print()
    print('State  Percent')
    print('-----  -------')
    for pair in pairs[-5:]:
        print(' {0:<2}     {1:>4.1f}%'.format(pair[0], pair[1] * 100))

    # Interactive querying.

    print()
    print('Enter the name of a commuting zone to find the chance that the')
    print('income of a child raised in that commuting zone will rise to')
    print('the top quintile if his or her parents are in the bottom quintile.')
    print('Commuting zone names have the form "Columbus, OH".\n')
    name = input('Commuting zone (or q to quit): ')
    while name != 'q' and name != 'Q':
        if name in data:
            mobility = data[name]
            if mobility == None:
                print('No data available for ' + name + '.')
            else:
                print('Percentage is {:<3.1f}%.'.format(mobility * 100))
        else:
            print('Commuting zone was not found.')
        name = input('Commuting zone (or q to quit): ')

main()
```